OPERATING ENVIRONMENTS FOR THE '90s

May/June 1990 Volume 8.3

Programmer's Journal™

.95 Canada $5.95

Undocumented DOS shell
page 28

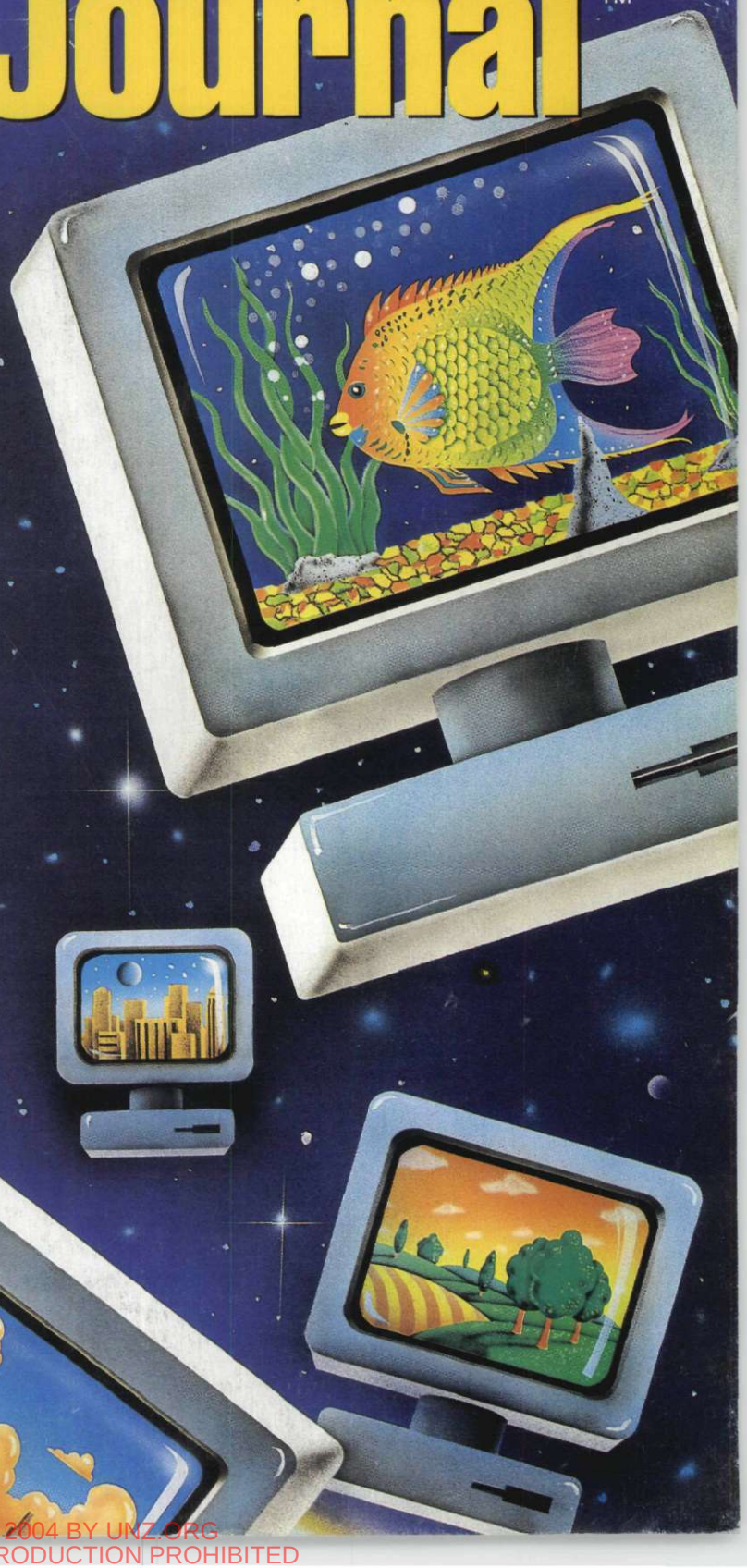
Graphics programming in Windows
page 40

Mousing around: messages in PM
page 48

ASM: advanced segmentation
page 64

Ellipses on the VGA
page 21

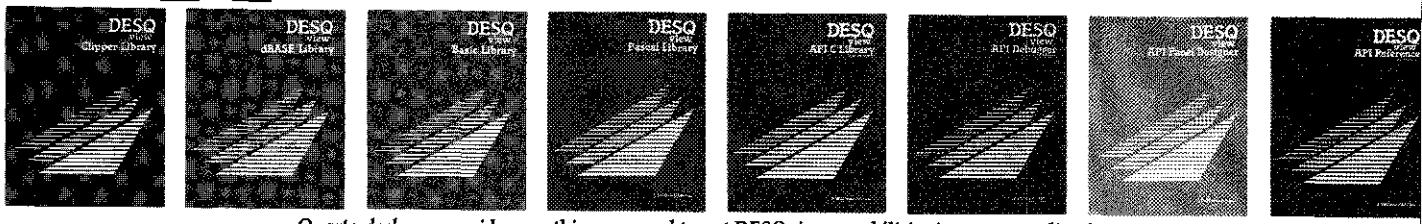
Up OOP alley in Pascal
page 79



05
0 74470 74130 5

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Now you can bring the power of DESQview to Clipper,™ dBASE™ and Basic.


DESQ view Clipper Library
DESQ view dBASE Library
DESQ view Basic Library
DESQ view Pascal Library
DESQ view API C Library
DESQ view API Debugger
DESQ view API Panel Designer
DESQ view API Reference

Quarterdeck can provide everything you need to put DESQview capabilities into your applications.

There's a growing realization among programmers that for many users, the ideal PC of the future doesn't look all that different from the PC of today. Experts agree that extended DOS will be the dominant operating system for the foreseeable future.

And DESQview is the best way to create powerful, multitasking solutions for the 30 million DOS PCs in use today.

The API Advantage

Programmers who take advantage of DESQview's API (Application Program Interface) get easy access to the powerful capabilities built into DESQview—multitasking, windowing, intertask communications, mailboxes, shared programs, memory management, data transfer, mousing, menu-building and context-sensitive help. In short, you get all the important benefits promised by OS/2. Without the overhead burden and performance trade-offs.

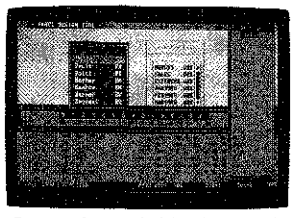
API capabilities are now available to Clipper, Basic and dBASE users as well as Pascal and C programmers.

Bells and Whistles

A program taking advantage of the DESQview API can spawn subtasks for performing background operations or new processes for loading and running other programs concurrently. It can schedule processing after an interval or at a certain time. It can use DESQview's intertask communications to rapidly exchange data between programs, share common code and data; or interrupt at critical events. It can use DESQview's menuing and mousing capabilities to create menus. And there's lots more it can do.

80386 Power

Programmers can take advantage of the 80386's protected mode for large programs, yet run on DOS and multitask in DESQview—side by side with other 80386 and DOS programs. The breakthroughs that make it possible: DOS Extenders from PharLap, Rational Systems, Eclipse and Oracle and DESQview support of these extenders.



Interactive tools like this Panel Designer make implementing the DESQview API a snap.

A Complete Interactive Toolkit —Including a Debugger

API Reference Manual The key to the power of the DESQview API, this manual contains all you need to know to write Assembly Language programs that take full advantage of DESQview's capabilities. And there's an 'include' file with symbols and macros to aid you in development.

API Clipper Library Add new power to client-server database applications. Define tasks that monitor or control any other application; run reports, sorts or numerical analyses in the foreground or background; create custom interfaces with control over keyboard, mouse and window functions. And it's all easy to accomplish because you can treat the entire set of over 300 object-oriented API functions as prewritten UDFs.

"With the DESQview API, you can create programs with abilities far beyond those of Clipper alone... obviously, the uses are limited by your imagination"
—Data Based Advisor, December 1989

You get the API Reference Manual and source code for the library with the Clipper Library package.

API dBASE Library All API functions are available within dBASE III and IV. We include source code for the library, the API Reference Manual and example programs.

API Basic Library The Basic library provides interfaces for the entire set of API functions. Included are the API Reference Manual, source code for the library and example programs.

API Pascal Library The Pascal library provides interfaces for all API functions. It supports Borland Turbo Pascal 4.0 and 5.0 compilers and Microsoft Quick Pascal. Included are the API Reference Manual, source code for the library and example programs.

API C Library You get C language interfaces for the entire set of API functions. It supports the Lattice™ C, Metaware™ C, Watcom C, Zortech C, C++, Microsoft® C and Turbo C compilers for all memory models. Included with the C Library package is the API Reference Manual and source code for the library.

API Debugger The DESQview API Debugger is an interactive tool enabling the API programmer to trace and single step through API calls from several concurrently running DESQview-specific programs. Trace information is reported symbolically along with the program counter,registers, and stack at the time of the call. Trace conditions can be set to report only calls of interest.

API Panel Designer This interactive tool helps you design windows, menus, help screens, error messages, and forms. It includes an editor that lets you build an image of your panel using simple commands to enter, edit, copy, and move text, as well as draw lines and boxes. You can then define the characteristics of the window that will contain the panel: its position, size, and title. Finally, you can specify the location and types of fields in the panel.

The Panel Designer automatically generates all the DESQview API data streams necessary to display and take input from your panel. These data streams may be grouped into panel libraries and stored on disk or as part of your program.

Some of the applications under development right now using DESQview API Tools: CAD, medical systems, insurance, 3270 network management, mainframe communications, real estate management, typesetting, point of sale, education, stock and commodity trading and online voting.

More Tools are Coming If you'd like to know all the things we're up to in the API Toolkit, plan to attend our API Conference this Summer. Call or write for details.

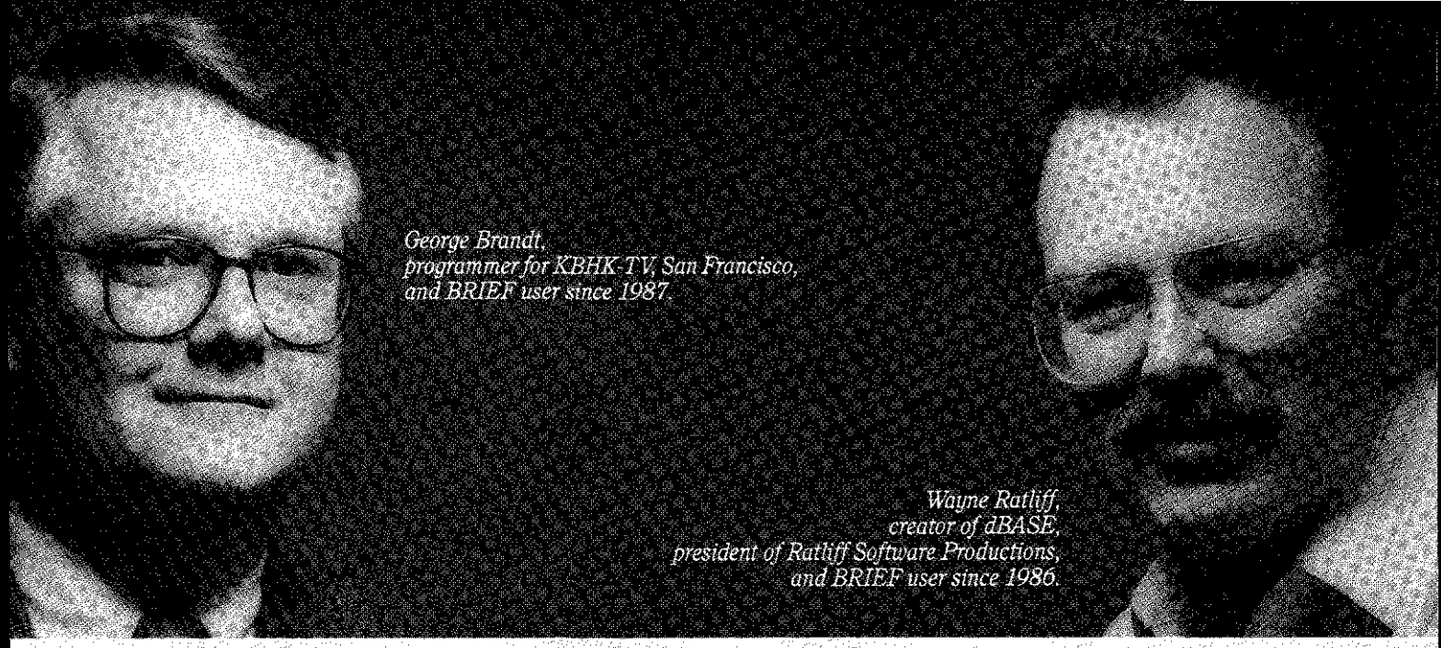
But for now, we can tell you that we are committed to adding tools as needed by our users. In the works, we have many new, exciting tools that will keep DESQview API users at the forefront of interface development.

Quarterdeck

Quarterdeck Office Systems, 150 Pico Blvd., Santa Monica, CA 90405 (213) 392-9851 Fax: (213) 399-3802

Trademarks: Clipper, Nantucket, dBASE, Ashton-Tate; OS/2, IBM; PharLap, Rational Systems, Eclipse, Eclipse; Oracle, Lattice, Metaware, Watcom, Zortech, Microsoft, Turbo C
©1990 Quarterdeck Office Systems

PRODUCED 2005 BY SOFTINFO.ORG.NZ
ELECTRONIC REPRODUCTION PROHIBITED



George Brandt,
programmer for KBHK-TV, San Francisco,
and BRIEF user since 1987.

Wayne Ratliff,
creator of dBASE,
president of Ratliff Software Productions,
and BRIEF user since 1986.

Of course they need completely different kinds of editors. That's why they both use BRIEF.

Let's face it. There's no such thing as two programmers who think alike. Or who work alike.

Because everybody's got their own style, their own habits. Their own quirky little requirements. As well they should. So, where are you going to find one editor that satisfies everyone? Simple. Look inside a BRIEF® box.

It's the right editor, right out of the box.

What you'll find inside is an enormously powerful, astonishingly easy, and highly-evolved program editor. One that works at blinding speed. And that's loaded with features designed to help you be more productive. One guaranteed to give you excellent results within one short hour of booting up the first time.

You'll find features like automatic indenting, infinite windowing, and powerful regular expression searching that goes well beyond the ordinary text-matching you're probably used to.

You'll also find the features that made BRIEF famous. Including our near-miraculous UNDO, that doesn't just un-delete your last keystroke. It actually reverses the effect of any command. So you can explore, experiment, even make gigantic mistakes with hideous consequences, and just "undo" them. And then continue to undo, backing-up through 300 steps.

What you want is what you get.

What you'll also find is an editor that delights in adapting itself to you. With the simple, menu-driven SETUP program, you can quickly customize indenting style, support for any of 36 compilers, keyboard configuration, rate of keyboard auto-repeat, and much more.

You can even edit BRIEF's macros to suit your taste. Or write your own in a snap.

Computer Language said that BRIEF is capable of "almost complete customization." And with new release 3.0, BRIEF is even more powerful and flexible.

For every programmer, it's a different BRIEF.

At last count, there were over 52,000 copies of BRIEF in the field, making life easier and working time more productive for all kinds of programmers—working in all kinds of languages. Programmers as different as Wayne Ratliff is from George Brandt. Which means there are over 52,000 different ways of working, and BRIEF fits them all.

Why not get BRIEF going to work for you? There's no risk. In fact, if you're not completely satisfied that BRIEF is exactly the editor you've been looking for, just return it within 30 days, and we'll return your money. No questions asked.

New BRIEF 3.0 highlights

- With our new C-like macro language, CBRIEF, C programmers can write macros on the fly and not have to switch between C and a different macro language.
- The new source level debugger works with both the original BRIEF syntax and the new CBRIEF macro language.
- Maximum lines per file has been increased to 4 billion.
- We've added to and improved template editing and smart indenting for ADA, Cobol, BASIC, FORTRAN, Modula-2, Pascal, and C.
- We've added editable, multiple keystroke macros that can be saved and restored.
- You can even save and restore your window set-ups.

Plus the features that made BRIEF famous.

- 300-level UNDO safety net
- Flexible windowing, unlimited quantity
- Unlimited number of files
- File size limited only by disk space
- Regular expression search and replace
- BRIEF's LISP-like macro language
- Toll-free tech support

International Distributors: England (0763) 73457 West Germany (069) 34 72 26 France (1) 46 22 9988 Australia (03) 663 6580 Denmark (42) 88 7249 Norway 03-89 22 40 Austria (1) 31 02031 Belgium (3) 829 0581 Israel 4 238757 Sweden (0) 300 70910 Switzerland (31) 586655

All product names are trademarks or registered trademarks of their respective owners. © 1989 Solution Systems

BRIEF 3.0

The Programmer's Editor

Solution Systems™
541 Main St., Suite 410
South Weymouth, MA 02190
FAX 617-337-7719

Try BRIEF at our risk. $199 plus $6 shipping (U.S. and Canada). 30 day money back guarantee.

Name _____
Company _____
Address _____
City _____ State _____ Zip _____
Phone _____

Paying by ☐ Check ☐ MC ☐ VISA
Card # _____ Exp. date _____
Signature _____
Disk size ☐ 5-1/4" ☐ 3-1/2" Format ☐ DOS ☐ OS/2

Mail this coupon or call toll-free today.

800-821-2492

Extension 445

Massachusetts & outside U.S. call 617-337-6963

Number 47 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Programmer's Journal™

May/June 1990 Volume 8.3

FEATURES

ON GRAPHICS **WHAT GOES AROUND** *By Michael Abrash* 21

Installment 18, in which Michael starts to work his VGA circle-drawing magic on the vastly more useful ellipse

ENVIRONMENTS **REENTERING THE DOS SHELL** *By Daniel E. Greenberg* 28

A detailed exploration of the undocumented INT 2Eh entry to the DOS shell—COMMAND.COM, and an XCALL utility to use these features

ENVIRONMENTS **ORDER FROM CHAOS** *By William S. Hall* 40

Our Windows wizard uses a cellular automaton as a vehicle to demonstrate graphics programming using Windows GDI

ENVIRONMENTS **MOUSE MESSAGES** *By Chip Little* 48

Efficient processing of mouse input in OS/2 Presentation Manager applications

OS **ACCESSING HIGH MEMORY** *By Larry Smith* 55

Exploring the extended-memory block-move features of AT-class machines

C **BENCHMARKING—FACT OR FICTION?** *By Rick Naro* 59

A close look at performance analysis and the issue of benchmarks illustrated using several C compilers

ASM **ADVANCED SEGMENTS** *By John Otken* 64

John explains 80x86 segmentation directives, how to use segments wisely in your assembly language programming, along with advanced run-time table-building techniques

PASCAL **LET'S LIST AGAIN, AND AGAIN, . . .** *By Tom Swan* 79

Tom continues work on his "View Anything" file viewer by creating some useful list and selection routines for files and directories using the object-oriented extensions in Turbo Pascal

COLUMNS

PJ FORUM **SOFTWARE DEVELOPMENT '90** *By William Roetzheim* 15

386 NOW **386 DEBUGGING** *By John Otken* 18

DEPARTMENTS

EDITOR'S NOTE 4

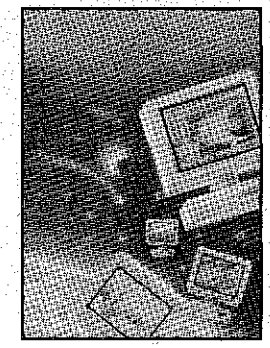
LETTERS 6

PJ PRESSROOM 90

PJ BILLBOARD 93

ADVERTISER INDEX 96

COVER



In this issue we examine the diverse and dynamic universe of operating environments. Let *PJ* be your command center from the high reaches of memory to orbits and reentrance.

Artist: Rob Schuster

Articles and letters in *Programmer's Journal* represent the views of the authors and not necessarily those of the publisher. Letters to the editor become the property of *Programmer's Journal* and may be edited for publication.

Code and listings in *Programmer's Journal* appear "as is," and *Programmer's Journal* makes no guarantee or warranty of their quality, performance, accuracy, or suitability for any particular purpose. *Programmer's Journal* assumes no responsibility or liability to any person or entity with respect to any damage caused or alleged to be caused by their use.

Advertising: For ad rates, deadlines, and placement, call Robert Tappan or Jay Moore at 800-234-0386, or write to *Programmer's Journal*, P.O. Box 70167, Eugene, OR 97401. ***Programmer's Journal*** is produced using the following software products: Ventura Publisher,™ Aldus PC PageMaker,® and Corel Draw. The PostScript™ files are output on a Linotronic™ 200.

Programmer's Journal (ISSN 0747-5861) is published bimonthly by Oakley Publishing Company, PO box 70167, Eugene, OR 97401. Second-class postage paid at Springfield, OR, and additional mailing offices. USPS # 000 293. **POSTMASTER:** Send address changes to *Programmer's Journal*, PO Box 70167, Eugene, OR 97401.

© 1990 OAKLEY PUBLISHING COMPANY

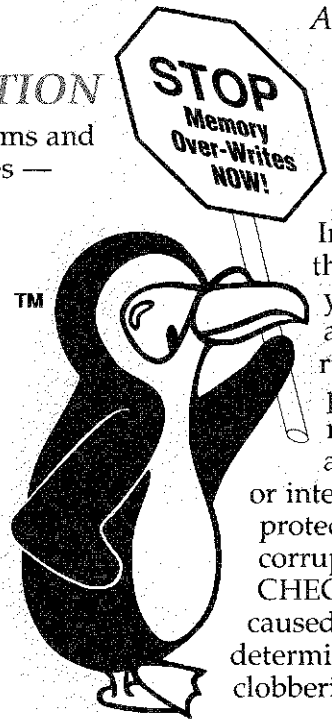
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Another New Product from the makers of Soft-ICE & MagicCV

ANNOUNCING BOUNDS-CHECKER

Gives you the protection of a protected operating system under MS-DOS.

Finds out-of-bounds memory accesses — AUTOMATICALLY.



AUTOMATIC PROTECTION

Flush out those Nasty pointer problems and other out-of-bounds memory accesses — AUTOMATICALLY.

Each time you make a change to a program, run BOUNDS-CHECKER while testing the new code. If you accidentally access out-of-bounds memory, BOUNDS- CHECKER will pop up displaying the offending SOURCE LINE. And your program runs at full speed.

Increase Productivity During Development, Improve Reliability During Q/A

You can run BOUNDS-CHECKER while testing your program. There are no additional steps to your testing cycle, but you can feel secure when the program has passed through BOUNDS-CHECKER with no reported problems.

Many over-write problems and other out-of-bounds memory accesses do NOT show up during normal testing. An out-of-bounds memory location may be modified, but that particular location doesn't happen to be important at the time. Once the program is in the field and a certain network is loaded or a certain T&SR or device driver is loaded, that memory location suddenly becomes very important... AND THE SYSTEM CRASHES.

You can prevent these problems by making BOUNDS-CHECKER a standard part of your testing procedure.

HOW IT WORKS

BOUNDS-CHECKER uses the 80386 virtual machine technology to provide real-time memory protection. In addition BOUNDS-CHECKER uses the symbolic information output by your compiler to differentiate CODE and DATA. When your program is running, BOUNDS-CHECKER protects the program's CODE and all memory outside your program. When an MS-DOS system call, BIOS call or interrupt occurs, BOUNDS-CHECKER protects the system software from corrupting your code. So, BOUNDS-CHECKER will not only detect problems caused by your program, it will also determine if a T&SR or other program is clobbering you.

Enormous Productivity Gain

Don't spend days or even weeks chasing down one of those subtle memory over-write problems. Use BOUNDS-CHECKER to pinpoint the problem in *seconds*.

Don't RISK a Recall

Don't take the unnecessary RISK that your program is unknowingly clobbering an out-of-bounds memory location. BOUNDS-CHECKER was developed to find these potential time bombs quickly, painlessly and AUTOMATICALLY!

$249.00

Requires 80386 PC.
MS-DOS is a trademark of Microsoft Corporation.

Nu-Mega TECHNOLOGIES

CALL TODAY (603) 888-2386 or FAX (603) 888-2465

P.O. BOX 7607 ■ NASHUA, NH ■ 03060-7607 ■ U.S.A.

Number 160 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

EDITOR'S NOTE

This issue of *PJ* focuses on PC operating environments, a competitive arena indeed. Among the major contenders along with MS-DOS, we have Windows running atop DOS, two flavors of OS/2 with Presentation Manager (version 1.2 for the 286 and version 2.0 for the 386), and various UNIX variants (mostly limited to 386 machines). To this list could be added other options such as Concurrent DOS (Digital Research), QNX (Quantum), NetWare (Novell), and other LAN systems. But in the PC marketplace one message is clear—lowly MS-DOS has incredible staying power and remains the operating system of choice. Suitably extended with 286 and 386 DOS Extenders, with Windows and other GUIs, DOS continues to prosper and thrive.

SMALL IS BEAUTIFUL

Others have spoken contemptuously of DOS as merely a program loader and not an operating system at all. And there is certainly a grain of truth in these views. However, it is precisely a result of DOS's small overhead and size (typically less than 60K of memory) and its limited command set (less to learn) that DOS still reigns supreme on the PC. The simple command line may be dated and more difficult to learn, but the modest impact of DOS allows other user interface options to be easily added. No doubt, the Apple Macintosh sports a more contemporary WIMP (Windows, Icon, Mouse, Pulldown menu) interface that may very well be easier to learn. But the drain on CPU time and memory resources required by MAC OS, Windows, OS/2 PM, and the UNIX GUI operating environments can appear to users as sluggish overall performance. I've always felt that UNIX suffered from too many commands for the uninitiated, ironically its attraction for the serious programmer.

MS-DOS began life as a modest 16-bit clone of CP/M. Both operating systems were compactly written in assembly language. Adding features from more powerful environments (tree-structured directories and command line redirection from UNIX, for example), DOS versions from 2.0 onward took on a different flavor but were still limited to the command line metaphor. DOS 4.x finally added menus directly. However, the small size of DOS and the speed of memory-mapped video allowed other text-windowing and graphics interfaces to be easily added. Contrast DOS with OS/2 or UNIX on the PC. Merely to install the initial version of OS/2 version 2.0 (for the 80386) requires a modest 6 megabytes of memory (although only 4 megs are needed after installation). DOS still runs on older, 256K XTs.

A PLETHORA OF NEW PROGRAMMING TOOLS

This should really be called "anticipatory" journalism. When writing these notes, I try to compose lines that just might be timely when read two months later. In the March editor notes written last December, I presumed that Microsoft C 6.0 would be available by March. Well, I was wrong. I presume that Microsoft C 6.0 might be released when you read this in May. Soon to follow are new releases from WATCOM (C/8.0 and FORTRAN for both 80X86 and 386 DOS Extenders), Borland (Debugger, C++, and Turbo Pascal 6.0), JPI (Pascal and Modula-2), and a host of others.

NEW ENVIRONMENTS, ANYONE?

Following on this issue's theme, we at *PJ* are introducing our own new "look and feel." Similar to GUIs competing in the PC and workstation marketplace, *PJ* now sports a new "visual metaphor," but with the same useful information and thoughtful material between the covers. We hope you find our new look more attractive and easier to use.

Steven

Programmer's Journal

Editor
M. Steven Baker
Technical Editor
Colin Elligsen
Managing Editor
Robert E. Wanetick
Contributing Editors
Michael Abrash
Frank D. Greco
Tom Swan

Authors
Michael Abrash
Daniel E. Greenberg
William S. Hall
Chip Little
Rick Naro
John Otken
William Roetzheim
Larry Smith
Tom Swan

Publisher
Elizabeth Oakley
Associate Publisher
Bobbi Sinyard
Production
Ralph Bentley
Illustration
Jim Carpenter
Advertising
Robert Tappan
Jay Moore
Circulation
Eric Malin
Fulfillment
Gary Hoyer
Accounting
Lori Fanslow
Kathy Schneider
Administrative Assistant
Sylvia Long

Business and Editorial Offices
Programmer's Journal
150 North 4th Street
Springfield, OR 97477
503-747-0800
800-234-0386
Fax: 503-746-0071

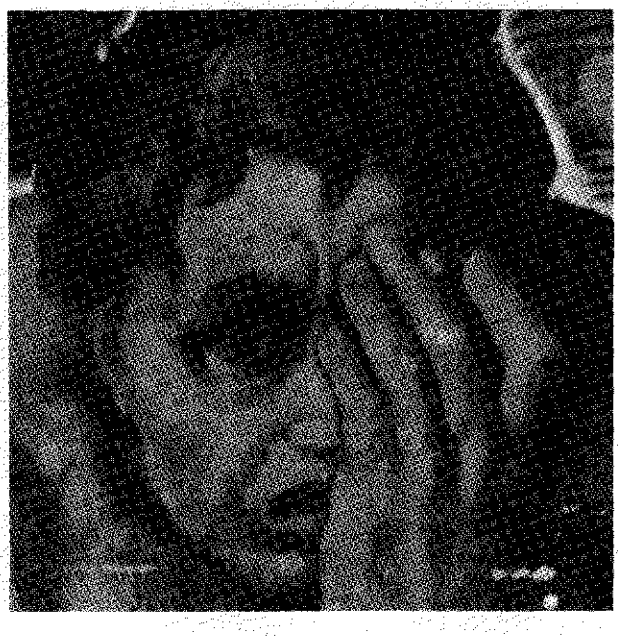
Subscriptions
1 year (6 issues) $24.00

Change of Address
Programmer's Journal
PO Box 70387
Eugene, OR 97401

© 1990 OAKLEY PUBLISHING COMPANY

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Death Taxes Software Piracy

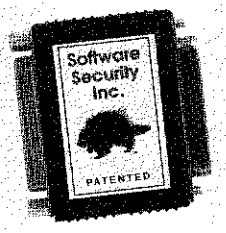


We can save you from one of them.

Sorry. Death we can't do anything about. As for taxes, when you use our product you'll probably wind up paying more. But software piracy: there we offer some help. Our family of software protection devices (dongles) have improved unit sales for over 2,000 companies around the world. Our products can be used in the MS-DOS, OS/2 and Macintosh environments.

Build Your Own Custom Protection Environment

Use our patented "dual-locking" ASIC chip as the basic building platform. Next, add options like: on-the-fly read/write memory, write-once or multiple-write locking codes, and encryption shells. Then add your own programming creativity to build a protection environment best suited to your product.



Users attach the device to their parallel port, and programs won't run without it. Back-up copies, hard disk and LAN operation are not interfered with.

Your Intellectual Property Belongs To You

And if you don't protect it, who will? Our products offer the most equitable way to protect your interests without sacrificing the rights of your customers. Call us today for information and demonstration units.

Software Security

1011 High Ridge Road - Stamford, CT 06905
1-800-333-0407 ext. 104
203-329-8870 Fax 203-329-7428 BBS 203-329-7253
AppleLink™ D2379

Macintosh is a trade mark of Apple Computer Inc., Activator, Mactivator are trade marks of Software Security, Inc. illustration: detail from Michelangelo's Last Judgement

Number 221 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LETTERS

ARE YOU LISTENING, MURRAY?

This is the first time I have ever written a letter in response to an article in a magazine. I have never had an article published in any magazine. Therefore, I'm ignorant of the feedback that one would receive by doing so. I sincerely hope that the feeling one would get is not "Gee, I hope someone read and maybe even put to use what I wrote." I can assure Murray L. Lesser that I, for one, read with interest his work entitled "Computing Time in C" [*PJ* 8.1].

Although I am not presently involved in a project that requires this kind of accuracy, I can't thank Mr. Lesser enough for including a method of patching DOS 3.3 to eliminate the "enhancement" the writers included. This has been a constant source of aggravation to me, as my situation requires me to temporarily change my system clock quite frequently. I was especially impressed that he took the time and space to explain a "quick and dirty" way and also the proper way to do this.

I want Mr. Lesser to know that although this type of thing is completely out of my line, I had absolutely no trouble following his instructions and was able to implement his changes on my first attempt. From this day on, every time I temporarily reset my system clock, I'll say a little thank you. He may not hear it, but rest assured, it'll be there.

Bart Hollis
President, Koalts Industries
New Britain, Connecticut

ASM MACRO LIBRARIES

In his Volume 8.1 column, John Otken says that "grafting block structure to your [ASM] programs with macros and preprocessors can easily ruin it." He is certainly entitled to his tastes, but I disagree. I have for several years used a system of my own design for structured ASM programming, which is quite reliable. It is cleanly integrated into the ASM language syntax, in effect making ASM a high-level block-structured language.

By eliminating the tedium of generating and tracking jump-target labels

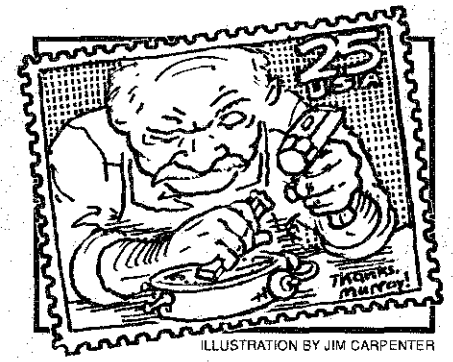


ILLUSTRATION BY JIM CARPENTER

by hand, it speeds initial program development and eases maintenance. In fact, I very rarely write a naked JMP opcode anymore—and usually think better of it after I do, replacing it with true structured code.

The system was originally developed as a set of macros for the M80 8080 assembler under CP/M. These were ported to MASM 1.0, but they slowed it down and ate up so much of the limited symbol space available that I replaced the macro implementation with a preprocessor program.

With the recent improvements in macro-assembler technology, I have recast the system as a macro library for MASM 5.1-compatible assemblers (including Turbo ASM and OPTASM). This allows me to debug the "high-level assembly" source directly using CodeView or Turbo Debugger. I have also developed a parallel macro library for the A86 shareware macro assembler by Eric Isaacson.

There are two structure forms, an extended **_if**/**_elseif** chain and a generalized **_loop**, which allows both **_while** at the top and **_until** at the bottom, as well as multiple internal **_break**s and **_again**s. For example:

```
_while    <call read_value>,nc
          ;CARRY => EOF
     _if       <cmp ax,100>,e
          ;if exactly 100
          call process_100
     _elseif   ,b
          ;flags from CMP AX,100
          call process_below_100
_break    ,c
          ;end loop on CARRY
     _else
          call process_above_100
_again    ,nc
          ;restart loop on NO CARRY
     _endif
_until    <tst go_flag>,z
          ;PROCESS_XXX said "stop!"
```

The general form is:

```
_IF  <operation>,condition
```

Operation is any valid opcode or macro, including arguments—and may in fact be empty if the CPU flags have been set by a previously executed operation—and *condition* is any of the jump-test conditions, such as NZ, AE, PO, etc. There are also **_nextcx** forms to take advantage of the CX-specific opcodes such as **loop** and **jcxz**.

As you may see in the fragment above, **_if**s and **_loop**s may be nested. Although it is not required, I usually indent the source lines to display the block structure of the program (**_break** and **_again** are indented to the level of their associated **_loop**, not to the level of the enclosing nested block, if any). I have a program, AB ("assembly beautifier," in homage to the UNIX cb, "C beautifier"), which does this automatically. It is itself written in "structured assembly" and serves as a good example of how the package is used.

Both the preprocessor and the macro implementations have been released into the public domain. The preprocessor is available on many bulletin boards as "strux.zip", and the macro source has been posted to the MASM conference on RelayNet. I plan to package the macro libraries (MASM and A86) with the source code for AB and post those for downloading as well.

Davidson Corry
Seattle, Washington

WHICH WAY DID IT GO?

On page 93 of *PJ* 8.1 was a short ASM program fragment for testing whether STDIN has been redirected. This program was contributed by Larry Smith of Friedland, West Germany, with a letter stating the reasons for wanting to make this test. It occurred to me that C programmers could do the same thing by using the handy **ioctl** function of the Turbo C library. I am enclosing a short C listing, "std_tst.c" [see sidebar], that will test whether either STDIN or STDOUT (caller's choice) has been redirected to a file. Also included is "ioctlt.c," a short program that calls "std_tst" to illustrate its use. These files were compiled using Turbo C 2.0 and linked to produce

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

IOCTLT.EXE on an AT clone using MS-DOS 3.2.

According to data in Ray Duncan's MS-DOS functions book [*The MS-DOS Encyclopedia*, Microsoft Press], the DOS function call 0x44 used internally by **ioctl** should work on DOS 2.0 and up. My two listings include "conio.h" so that the Turbo C **cprintf** function can be used to output to the screen even when STDOUT has been redirected.

To show that the **std_tst** function works, invoke IOCTLT.EXE from the DOS command line without redirection; it should report that both STDIN and STDOUT are devices. Redirect either or both when invoking IOCTLT.EXE, and the program should report the redirection. If you redirect STDIN to take input from a file that does not exist, DOS will generate an error. So, to test input redirection, be sure the input file exists.

The library of Microsoft C 5.1 does not include the **ioctl** function, but there is another function, **fstat**, which I believe could be used in place of **ioctl** with a few changes to the listing. I haven't worked out the details, but it should be straightforward. Likewise, my old Lattice C compiler version 3.1 has a function, **getfc**, which could be used with a few changes. I haven't checked the library of any other C compiler.

Keep up the good work.

Bernard H. Robinson, Jr.
DeBary, Florida

MAKING UP

John Otken ("A Library System for Assembly," *PJ* Vol. 7.5) points out two problems in his Microsoft/Borland–based method for maintaining a library. First, the OBJ files must remain in the directory at all times; second, the assembler is invoked on a per-module basis.

Both problems are solved in John's SLR Systems–based method, using OPTASM's and OPTLIB's built-in MAKE capabilities. Ken Green ("Keeping Up to Date," Letters, *PJ* Vol. 8.1) incorporates a sort of MAKE capability into his ASMLIST.EXE, taking care of the first problem, and uses Borland's TASM to handle the second problem.

Both methods, however, have a third imperfection. Assume, for example, that you modify two modules, FOO.ASM and BAR.ASM, before you run your library-maintenance system. Both systems will attempt to reassemble FOO and BAR. Now, assume that FOO assembles correctly, while BAR contains errors. Both systems will detect that BAR failed the assembly step and therefore will not update the library. So far so good.

You fix BAR and rerun the library maintenance system. Both systems will reassemble not only BAR but FOO as well. Why? Because FOO.ASM remains newer than the library! Only FOO.OBJ, which is ignored at this point by both systems, is newer.

The solution to this problem involves yet another MAKE step. Any source file whose OBJ file exists and is newer then the source file should be excluded from the "needasm" list.

CODE FROM B. H. ROBINSON'S LETTER

LISTING 1

```
/*   ioctlt.c  tests  std_tst function
*    Written by: B. H. Robinson Jan 21, 1990
*       Version: 001
*    Compile on Turbo C
#include <stdio.h>
#include <io.h>
#include <conio.h>

int std_tst( int channel );
void check( int ch );

char *s_channel[ ] = {"STDIN" ,
                      "STDOUT" ,
                      " BAD CHANNEL " };

int main()
{
    check( 0 );   /* Check stdin channel  */
    check( 1 );   /* Check stdout channel */
    return 0;
}

void check( int ch )
{
    int stat;

    if( (ch != 0) && (ch != 1) )
        ch = 2;
    stat = std_tst( ch );
    switch( stat ){
        case -1:
           cprintf("Error in st_tst for %s.\n\r", s_channel[ ch ] );
            break;
        case 0:
            cprintf("%s is a file.\n\r", s_channel[ ch ]);
            break;
        case 1:
            cprintf("%s is a device.\n\r", s_channel[ ch ]);
            break;
        default:
            break;
    }
}
```

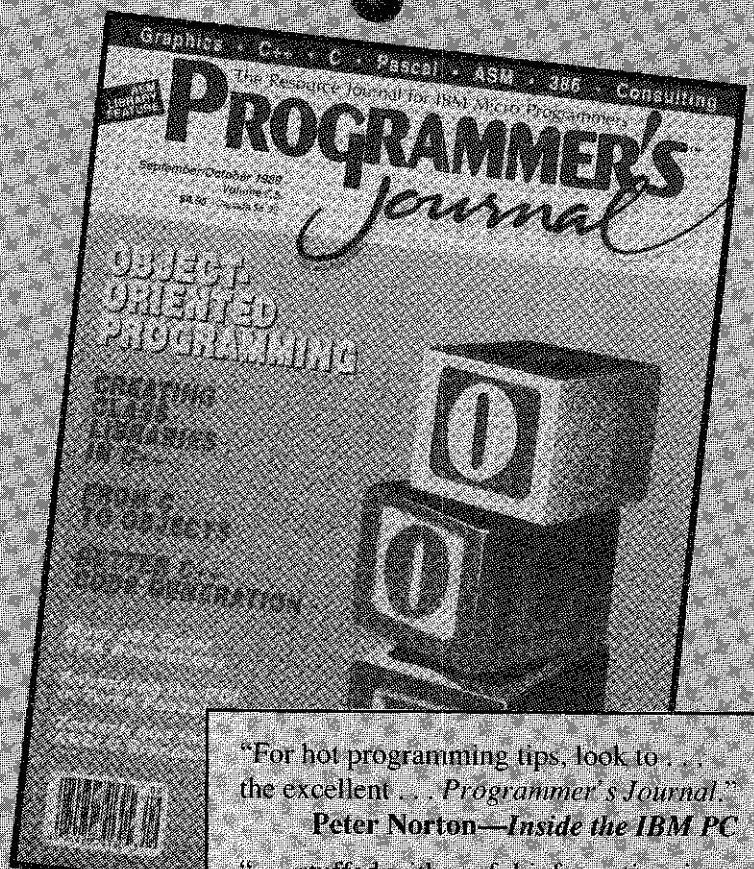
LISTING 2

```
/*   std_tst.c   std_tst function
*    Tests whether stdin or stdout has been
*    redirected to a file
*    Written by: B. H. Robinson
*       Written: Jan 21, 1990
*       Version: 001
*    The parameter "channel" must be 0 to test stdin or
*    1 to test stdout. stdin_tst returns 0 if the channel
*    is redirected to a file, 1 if it is a device or -1
*    if there is an error in execution of std_tst.
#include <stdio.h>
#include <io.h>
#include <conio.h>

int std_tst( int channel )
{
    int stat;

    if( (channel != 0) && (channel != 1) ) {
        cprintf( "Incorrect parameter in call to std_tst.\n\r");
        return -1;
    }
    stat = ioctl( channel, 0 );
    if( stat == -1 )
        return -1;
    if( (stat & 0x80) == 128 )
        return 1;
    else
        return 0;
}
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

"...love this magazine!"*

Industry experts agree that when it comes to a truly useful programming magazine, ***Programmer's Journal*** just can't be beat. Recognized by the **Computer Press Association** in 1989 for excellence in journalism, ***PJ*** is ". . . full of practical information and guidance presented with superb clarity and perspective." Try ***PJ*** and judge for yourself. It's easy—**just call toll free.** Start receiving 6 issues a year for only $19.95.



To subscribe call toll free:

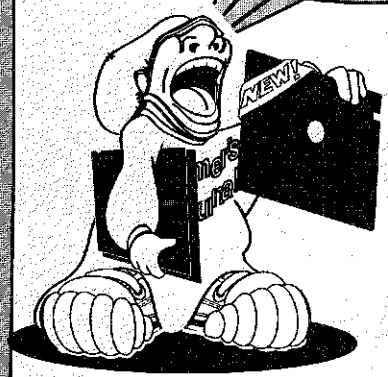
800-234-0386

"For hot programming tips, look to . . . the excellent . . . *Programmer's Journal*."
Peter Norton—*Inside the IBM PC*

". . . stuffed with useful information, it definitely deserves a look."
Ray Duncan—*Dr. Dobb's Journal*

* "I really love this magazine!"
Paul Mace—Paul Mace Software

PROGRAMMER'S JOURNAL
DiskScription
Six *PJ* Code Disks now only—
$39.95
1 Year (6 issues)



All Our Code— On Disk

Typing in long code listings uses up a lot of valuable time. And calling up bulletin boards is not exactly a free ride. So what does the modern programmer do? Get a ***PJ* Code DiskScription,** of course!

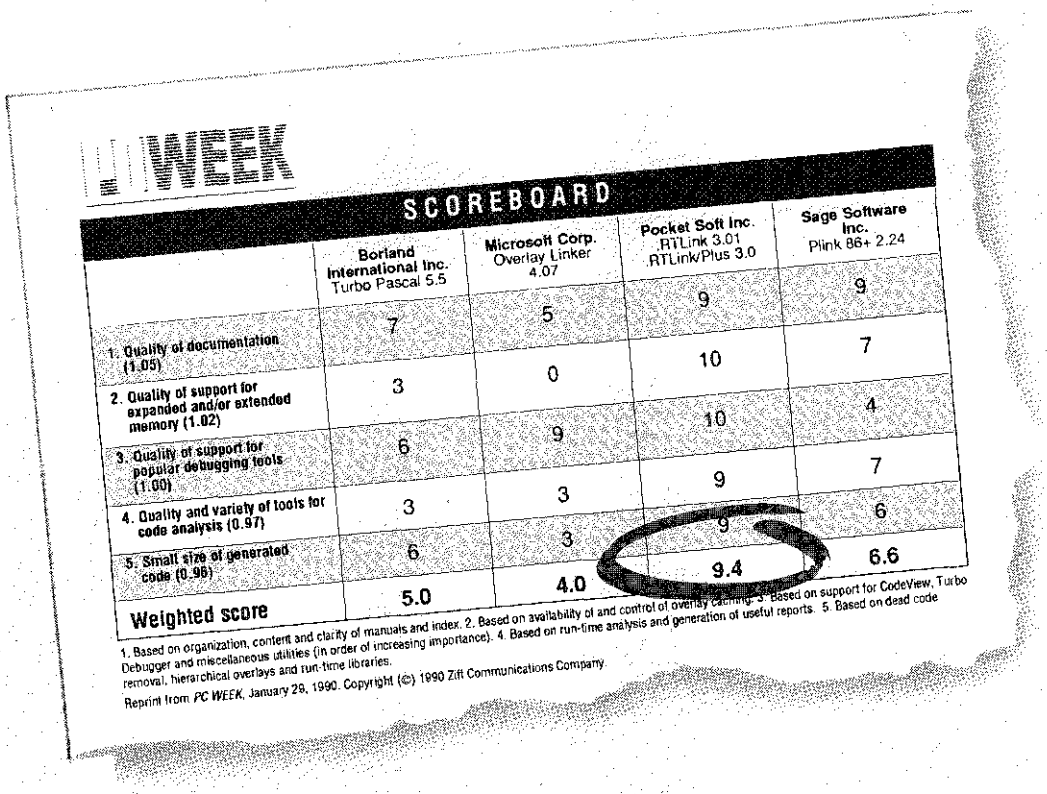
Each disk contains all of the program listings published in ***PJ*** plus .EXE files, .OBJ files, benchmark source files and more! If purchased separately the regular price for a year's worth of ***PJ*** code (6 disks) would be $60.00. **You can save yourself some time, effort and over $20.00 by ordering your *PJ* Code DiskScription today!**

Guarantee: If you are dissatisfied with your PJ Code DiskScription for any reason, we will gladly send you a full refund on all unmailed diskettes.

All *PJ* Code DiskScriptions must be prepaid. Return the card on the opposite page with payment. Or, for faster service with your VISA MC call toll-free: 800-234-0386

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

.RTLink/Plus™



PC WEEK

SCOREBOARD

	Borland International Inc. Turbo Pascal 5.5	Microsoft Corp. Overlay Linker 4.07	Pocket Soft Inc. .RTLink 3.01 .RTLink/Plus 3.0	Sage Software Inc. Plink 86+ 2.24
1. Quality of documentation (1.05)	7	5	9	9
2. Quality of support for expanded and/or extended memory (1.02)	3	0	10	7
3. Quality of support for popular debugging tools (1.00)	6	9	10	4
4. Quality and variety of tools for code analysis (0.97)	3	3	9	7
5. Small size of generated code (0.96)	6	3	9	6
Weighted score	5.0	4.0	9.4	6.6

1. Based on organization, content and clarity of manuals and index. 2. Based on availability of and control of overlay caching. 3. Based on support for CodeView, Turbo Debugger and miscellaneous utilities (in order of increasing importance). 4. Based on run-time analysis and generation of useful reports. 5. Based on dead code removal, hierarchical overlays and run-time libraries.

Reprint from *PC WEEK*, January 29, 1990. Copyright (©) 1990 Ziff Communications Company.

The clear winner against first rate competition.

.RTLink/Plus™ was the clear winner in *PC Week*'s review of overlay linkers. Overlay linkers are used to reduce MS-DOS program memory image size.

PC Week pointed out that .RTLink/Plus added several tricks of its own to expand link time technology to allowing programs to execute in less memory, improve performance, save development time and save substantial amounts of disk space in shipping media and on target machines.

.RTLink/Plus is the only multiple and nested overlay linker which fully supports Microsoft's CodeView© Debugger.

Get Pocket Soft's .RTLink/Plus and get new features which extend any MS-DOS compiler to a new and higher level of productivity.

Send us your name and fax number (or address if you prefer) and we will send you technical specifications on .RTLink/Plus and a reprint of the *PC Week* review.

.RTLink/Plus– Clearly the Best

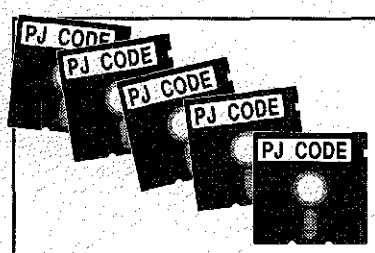
Pocket Soft, Inc.

Fax **(713) 460-2651**
Phone **(713) 460-5600**

P.O. Box 821049
Houston, TX 77282

Pocket Soft is an authorized licensee of Microsoft®, CodeView® information. "Microsoft" and "CodeView" are registered trademarks of Microsoft Corporation. .RTLINK, .RTLINK/PLUS and Pocket Soft, Inc are trademarks of Pocket Soft, Inc. Turbo Pascal is a registered trademark of Borland International, Inc. Plink86+ is a registered trademark of Phoenix Technologies, LTD.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



On disk from:
Programmer's Journal
P. O. Box 70167
Eugene, OR 97401-0110
Call toll free:
800-234-0386
In Oregon call:
503-747-0800
All Code Disks $10.00 each.
To order, see ad on page 88.
PJ Code DiskScriptions
are available for only $39.95.
To order, see ad on page 8.

On BBS from:

Buffer Board
538 Cardinal Way
Springfield, OR 97477
(Voice) 503-741-1414
(Data) 503-747-7636

EDI
1332 Hermosa Ave. #7
Hermosa Beach, CA 90254
213-374-3478
Call for your local access number.

Generation Five Inc.
12061 Tech Road
Silver Spring, MD 20904-9998
(Voice) 301-680-3600
(Data) 301-680-9300
Accessible via PC Pursuit,
Washington, D.C.
From Files submenu, select
"P" for Personal Downloads.

Gilmore Systems of Southern California
5447 Indian Hills Drive
Simi Valley, CA 93063
(Voice) 805-582-1360
(Data) 805-582-9306

Ithaca Street BBS
1145 Ithaca
Boulder, CO 80303
(Voice) 303-494-8865
(Data) 303-494-8868

Logical Alternatives Incorporated
Calder Square,
P.O. Box 10674
State College, PA 16805-0674
(Voice) 814-234-8088
(Data) 814-234-1881

Mnematics Videotex
P.O. Box 19
Sparkill, NY 10976-0019
(Voice) 800-322-3633
(Voice) 914-365-0184 in N.Y.
Mention *PJ* when you call.

PCS-49 on CompuServe
Section 10
Sponsored by:
MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170
(Voice) 703-450-4181

The Silicon Zone
820 Tivoli Circle #203
Deerfield Beach, FL 33411
(Voice) 305-360-9362
(Data) 305-360-9376

What you have to do, in essence, is perform two "nested" MAKE functions. You could run two MAKEs, the first creating a makefile for the second.

There is, however, a MAKE utility currently on the market, appropriately called VersiMAKE, capable of handling a variety of similar problems in day-to-day software-maintenance situations. VersiMAKE does not support the syntax conventions employed by UNIX MAKE and its many clones. Most of the information contained in UNIX-style "Dependency Files" is simply not needed by VersiMAKE. VersiMAKE statements are written in a structured control language that is both powerful and easy to use. The syntax is similar to that used by the C programming language.

The listing shows a library-maintenance system that incorporates John's and Ken's (and some of my own) ideas into a single VersiMAKE makefile. A free demonstration disk, containing the fully functional, capacity-limited version of VersiMAKE and a complimentary copy of VersiTOUCH can be obtained by writing to VersiMAKE Customer Support, Summit Information Systems, Inc., 496 N. Kings Hwy., Suite 130, Cherry Hill, NJ 08034.

Jan Fertig
Summit Information Systems, Inc.
Cherry Hill, New Jersey

We at *Programmer's Journal* like to hear from our readers. Please address correspondence to Editor, *Programmer's Journal*, 150 N. 4th Street, Springfield, OR 97477. You can also contact us via fax (503-746-0071), CompuServe (user ID 76676,427), or MCI Mail (user ID 336-4248). All letters become the property of *Programmer's Journal* and may be edited for publication.

CODE FROM JAN FERTIG'S LETTER

```
# LIBRARY.PRM
#    Updates a library file of ASM routines

define(libname, "LIBRARY")

# 1. Make list1, a list of all source files
#    in the current directory
#    whose object files must be refreshed in the library

define(list1, "")
automake((*.ASM),
         (),
         [libname].LIB,
         append(list1, "[dne] "))

# 2. Make list2, a list of all source files
#    out of those present in list1 (if any)
#    whose object files must be recreated

define(list2, "")
automake(([list1]),
         (),
         [rn].OBJ,
         append(list2, "[dne] "))

# 3. Recreate object files from source files in list2

repeat(([list2]),
       detach("MASM /Mx [rne];"))
wait()
exit(1)

# 4. Refresh object files in the library

repeat((*.OBJ),
       run("LIB [libname].LIB -+ [rne];")
       exit(1))

# 5. Update library listing and cleanup

make([libname].LIB,
     [libname].LST,
     run("LIB [dne], [tne]")
     dos("DEL *.OBJ"))
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Unleash Your 286, 386 or 486!

Whether you are running 1-2-3 or AutoCAD, or porting a FORTRAN application, MicroWay has the solutions that will both speed up your system and simplify your development efforts. These include inexpensive 386 accelerators, a 486 motherboard that has twice the throughput of a VAX 8650 when equipped with a Weitek 4167, a family of globally optimized 386 and 486 compilers and the most up to date numerics on the market. These include Intel, Weitek, and Cyrix coprocessors, plus Weitek daughterboards and a Weitek Micro Channel card for the model 70 and 80 — all of which comes with the finest technical support in the industry.

Milt Capsimolis of Ithaca Software, developer of HOOPS, the highly regarded 3D, object-oriented graphics library reports: "We ported a huge C library — well over 100,000 lines — without a hitch, in less than a day! ...We liked the enormous advantage NDP C-386 offers through its support of the Weitek coprocessor."

Fred Ziegler of Aspen Tech in Cambridge, MA: "I ported 900,000 lines of source in two weeks with NDP Fortran-386 without a single problem!" Aspen Tech's Chemical Modeling System is in use on mainframes worldwide and is probably the largest application to ever run on an Intel processor.

Our compilers come with the features you need to simplify porting to the 32-bit mode of the 386, including a 99% VAX VMS compatible FORTRAN and a dual dialect C which is UNIX System V and ANSI compatible. Also included is a library of 135 character and pixel oriented graphics routines that automatically detect and support the full range of PC display adapters. Plus we carry a full line of third party libraries and utilities that were ported with our languages.
For information about numeric coprocessor performance, call for your free copy of an article by Stephen Fried, "The State of PC Numerics in 1990". **For more information, please call our Technical Support Dept. at (508) 746-7341.**

386 Compilers and Tools

Our NDP family of compilers generate globally optimized, mainframe quality code that runs on the 386 or 486 in protected mode under UNIX, XENIX, or extended DOS. The compilers address 4 gigabytes of memory while supporting the 80287, 80387, Weitek, and Cyrix coprocessors. Applications can mix code from all three compilers and assembly language. The DOS versions allow the user to write his own numeric error handlers and interface 386 real mode programs from protected mode.

NDP Fortran-386™ is a full F77 with F66 and DOD extensions that is 99% VMS compatible.

NDP C-386™ runs as a full K&R C with MS extensions or as an ANSI compiler.

NDP Pascal-386™ is a full ANSI/IEEE Pascal, with extensions from C and BSD 4.2 Pascal.

DOS 386 versions (NDP tools included) $895
DOS 486 versions (NDP tools included) $1195
UNIX/XENIX versions $795
NDP VMM virtual memory manager .$295
Eclipse or Phar Lap Tools $495
NDP Link - Incremental Linker $295

NDP C++ Version 1.2 ..... $295
NDP Windows™ Library: $125, C Source: $250
NDP HOOPS™ $795
NDP Plot™ $325
NDP/FFT™ NDP or 80x87 version ... ea. $250
NDP to HALO '88 Graphics Interface $100
NDP NAG™ — The NAG Workstation library is a subset of the NAG mainframe libraries. It contains a library of 172 routines designed to solve differential equations and eigenvalue problems, perform matrix operations, fit curves, do statistics and regression analysis, generate random numbers, and compute special functions and integrals: $795, Weitek version: $995

NDP Fortran-486™, NDP C-486™, and **NDP Pascal-486™** are new releases of our NDP compiler family that have been optimized for the 486. Driven by NDP Fortran-486, the Number Smasher 486, running with a 4167, delivers over 10 million floating point operations per second! The compilers are $1195 each.

PC Accelerators

SuperCACHE-286 12 MHz $399
FastCACHE-286 12 MHz $299

Parallel Processing

MicroWay's IBM compatible Monoputer, Quadputer, Videoputer, and Linkputer boards work together using Inmos transputers to provide expandable, plug-in mainframe performance for your desktop PC.

Monoputer™ — Includes one T800 and up to 16 meg of RAM for parallel code development. The 4 MWhetstones T800 makes it the ideal FORTRAN engine for cost-effective execution of your mainframe programs... from $1295

Quadputer™ — This board for the AT or 386 can be purchased with 1 to 4 transputers and 1 or 4 meg of memory per transputer. Two or more Quadputers can be linked together to build networks of up to 100 or more transputers providing mainframe power ..... from $1995

Linkputer™ — Links up to 8 boards to provide dynamic transputer topologies $1500

Transputer Compilers and Utilities
These parallel languages are designed for use with the Monoputer, Quadputer and Videoputer.
Logical Systems Parallel C $595
3L Parallel C, FORTRAN, or Pascal .. $895
TBUG — debugger for 3L $330
Parsec Parallel C/dynamic $1500
ParaSoft EXPRESS Package — Includes transputer communications libraries, parallel code development library, C source level debugger, and system performance monitor $1500
Helios PC/s $1250
occam 2 Toolset $1500
Nexis Windows File Server — Lets you run parallel applications under the Microsoft Windows environment $495
T800/NAG™ — Port of the complete NAG mainframe library. Contains 268 functions: $2750

486 Your PC!

Number Smasher-486™ is a 25 MHz replacement motherboard for ATs and 80386s. This motherboard supports an optional Weitek 4167 numeric coprocessor and up to 16 megabytes of memory. It is ideal for applications such as CAD/CAE workstations, statistical analysis, linear programming, and multi-tasking operating environments, such as UNIX, XENIX, and OS/2. It supports the Phoenix, AMI, or Award BIOS. The Number Smasher-486 with 0K is priced at $3995.

Math Coprocessors

WEITEK

4167-25 (Now in stock) $1395
mW1167 Micro Channel-16/20 .. from $995
mW3167 Micro Channel-25/33 .. from $1495
3167-20 ..... $795
3167-25 ..... $995
3167-33 ..... $1295
mW3167/80387 Board $200
mW1167™ and **mW3167™** coprocessor boards are built at MicroWay using Weitek components. Each includes an 80387 socket.

INTEL

8087	$84	**80287-2**	$120
80287-8 ...	$195	**80287-10**	$220
80387-16 ..	$330	**80387-16SX** ..	$310
80387-20 ..	$400	**80387-25**	$510
80C287A ..	$280	**80387-33**	$625

287Turbo-20™ This coprocessor board runs a specially qualified Intel CMOS 80287 at 20 MHz regardless of the main CPU speed .. $450
RAMpak™ - one or four meg 32-bit memory expansion module for Compaq Deskpro 386 20/25 One meg .. $240, Four meg .. $700

CYRIX

Cyrix CX83D87 FasMath™ — Fastest 80-bit Intel compatible coprocessor. Performs transcendentals 3 times faster than the 80387.
20 MHz: $400 **25 MHz:** $510 **33 MHz:** $625

386 Your AT!

NUMBER SMASHER-386™ — A full-sized card that replaces the 80286 microprocessor on your IBM AT or compatible motherboard with an 80386 that runs at 20 or 25 MHz. It runs numerically intensive applications up to a factor of 60 times faster, while maintaining full hardware and software compatibility when running all 386 applications. Includes sockets to optionally add up to 8 megabytes of 32-bit memory, an Intel 80387, Weitek, or Cyrix numeric coprocessor, and 64K or 256K of high speed cache memory .... from $895

FASTCache-SX™ — The most cost effective accelerator we have ever manufactured. Plugs into the 286 socket, speeding up all applications by a factor of 2 to 4. Runs all 386 applications. Features an 80386-SX (16 or 20 MHz), a 4-way 32K cache, expandable to 64K, and a math coprocessor socket. 16 MHz:$495 20 MHz:$595

MicroWay

World Leader in PC Numerics

Corporate Headquarters: P.O. Box 79, Kingston, MA 02364 USA (508) 746-7341
32 High St., Kingston-Upon-Thames, U.K., 081-541-5466 Germany 069-75-2023
USA FAX 508-746-4678 Italy 02-74.90.749 Holland 40 836455 Japan 81 3 222 0544

Number 104 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

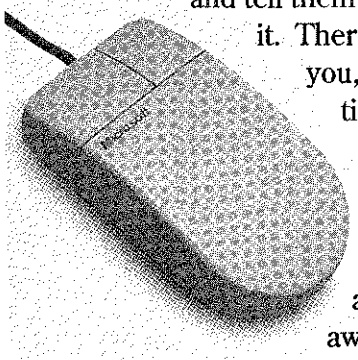
This team created your next generation editor. Tell us if you agree.

From left to right: ***Jim Besemer****, Project Manager, Specification & Design, User Interface;* ***Skip Russell****, System Primitives, Function Library, Error Recovery, Operating System Interface, Configuration and Setup;* ***Pat Thompson****, Extension Language/Compiler and Debugger;* ***Robynne Major-Jones****,* ***Howard Mercier*** *(not shown) Quality Assurance and Testing;* ***Eric Johnson****, Editor Core, Help Facility;* ***Milo Wadlin****, User Documentation.*

For two years this team worked in secret to create the world's best editor. We believe they succeeded magnificently. Now we'd like to know what you think.

New development environments and the size and complexity of projects make your 80's editor inefficient and obsolete. So we built you an editor for the 90's.

Try the Sage Professional Editor for thirty days with no risk. Pound on it. Compare it to any editor you've ever used. Then call our team at 1-800-547-4000 and tell them how you like it. There's no risk to you, no obligation. Not much risk for us either. We won't be able to pry it away from you.



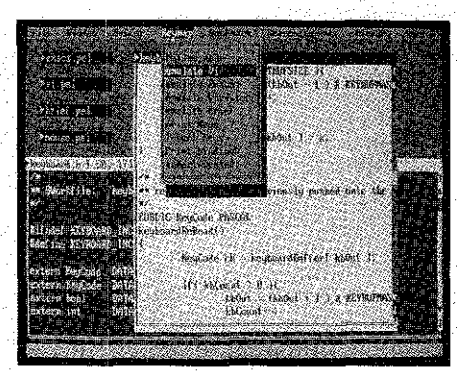
Works with or without a mouse.
Packaged with or without a Microsoft® Mouse.

Announcing the Sage Professional Editor

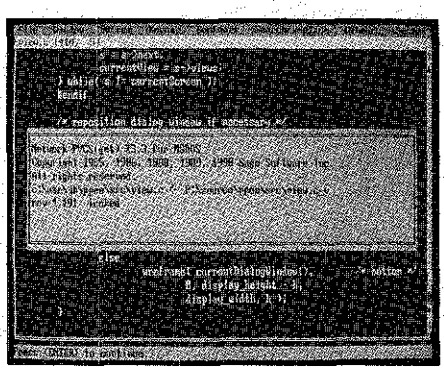
Right out of the box you'll be more productive with our editor than any you use today. The instant installation, elegant mouse support, advanced user interface, and point-and-shoot help get you running immediately. If you prefer the commands and keystrokes of a popular editor, our turnkey emulations duplicate them precisely, and you still gain the Sage Professional Editor's advanced features, windowing capabilities and powerful engine.

Later you'll make this editor truly yours by configuring the interface as you prefer. Every feature can be turned on or off as you like - from a clean screen, to tiled windows, to overlapping windows in various colors, pull-down menus, rulers, scroll bars, and line numbers - choose any or all and place them as you like.

Use the editor with or without a mouse - all functions are available without lifting your fingers from the keyboard. But the click-select windows, scroll bars, zoom, shrink-to-icon, block text selects, and other speedy mouse actions make editing extremely efficient.



Make our interface what you prefer, from clean screen to multi-window with drop-down menus and icons.



The editor environment provides seamless integration to the Polytron Version Control System (PVCS) or any other tool you care to connect.

Sage Software, Inc., 1700 NW 167th Place, Beaverton, OR 97006, (503) 645-1150, Fax (503) 645-4576, Int'l. (301) 230-3307, Int'l Fax (301) 984-3047.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Pop open the DOS window and the editor shrinks to just 4k. So you can back-task to compilers and other tools without leaving the editor.

This package is stuffed with value. It includes MS-DOS, OS/2 and Dual Mode versions on both 5 1/4" and 3 1/2" disks, templates for popular languages, and you can buy it with or without a bundled Microsoft® Mouse.

But the reason you will become addicted to the Sage Professional Editor is the power behind that pretty face. We looked under the hood of other editors and found wimpy editing engines dressed up with all sorts of bells and whistles. Unacceptable. We wanted an industrial strength engine. So we started with a powerhouse virtual memory system that allows us to edit huge files (up to 100MEG) in as many as 256 windows - over two billion lines. It makes maximum use of available memory (including EMS) and uses advanced processes to safely minimize file I/O. All higher level services use this powerful VM scheme. Consequently, there are no size constraints on the macro library and no limit to Undo/Redo. You can have 1000 bookmarks, anchors and saved positions per buffer.

TurnKey Emulation Of

Vi | Brief | Wordstar | Epsilon/EMACS | Others

Sage Software, Inc. has acquired Polytron, the company that brought you PVCS and PolyMake. Now this group proudly introduces their Professional Editor under the Sage banner.

DEVELOPMENT POLYTRON TOOLS

"When I saw how well my mouse works with this new editor I had to try it. I became a convert when I saw the power of the extension language. This editor can be whatever I need it to be. I couldn't give it up."

Terry Hamm,
ImageBuilder Software

And then there's the extension language. The Sage Professional Editor uses a C-like extension language (100 percent PolyAWK compatible) and compiler/debugger that programmers find immediately intuitive. The function library gives access to nearly everything. From mouse click to menu, from word processing to external hooks. You can build the environment you need or want with the editor as the front end to your favorite tools. The seamless integration of the Polytron Version Control System (PVCS) is a sterling example of how cleanly you can hook external programs.

Emulations of Vi, Brief, EMACS, and WordStar, were written with the extension language. The source code for each is included. If you are addicted to some obscure editor you can modify a copy of emulator source and quickly make your own. Source for the menuing system, mouse interface, help and many other services is also included, so you can deeply modify the editor to be exactly what you want. Brief users who have a library of macros can use our translator to recompile them into our more useful and powerful C-like extension language.

It's easy to make the Sage Professional Editor a standard for your team or even the entire company:

1. Turn-key emulation means every programmer can use it immediately. Macros written to aid users apply to all emulations.

2. Provides a framework for integrating all your development tools into a single, consistent operating environment.

3. MS-DOS, OS/2 & Dual Mode versions are all included in the same box so *your* team can move into the 90's with the same editor.

4. Licenses are available for you, your team, division or company. The initial cost is low, and you continue to save as you add users.

MS-DOS | OS/2 | Dual Mode — All in One Package

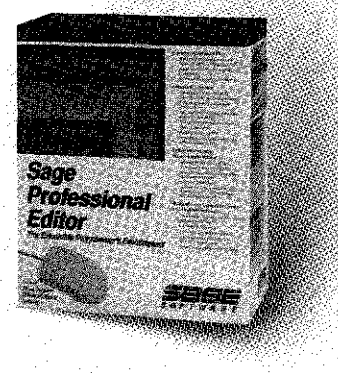
$295 U. S. List Prices. Single User Version.

$395 Includes a Microsoft® Mouse.

Specify serial, bus or PS/2 mouse. A 5-User License (without mice) is only $1,032.50. Call for prices on larger licenses. **Visa, MC** & P.O. orders:

1-800-547-4000

30 Day Money Back Guarantee



SAGE SOFTWARE

Number 26 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FINALLY
THE IDEAL "C" SEMINAR!

Within one week, you can increase the value of your employees by adding "C" to your company's programming language vocabulary!

CONVENIENT COST-EFFICIENT "C" TRAINING

Thousands of people have been trained and they have given CLA courses consistently superior reviews with our competitively priced courses. Become proficient in ANSI "C" in one week.

FLEXIBLE SCHEDULING

Public classes are offered monthly in our Collingswood office, convenient to Philadelphia. Classes can also be scheduled at your company site.

FOR FAST INFORMATION
CALL TOLL FREE
1-800-842-9667
and we will FAX or mail you the latest brochures

CLA INC
COMPUTER LANGUAGE ARTS

704 Haddon Avenue; Collingswood, NJ 08108

Number **300** on the Reader Service Card

Data Junction®
converts data files!

***Any* ASCII • EBCDIC • c-tree • Packed data**
Binary • dBASE • R:BASE • Paradox • Xdb
Excel • 1-2-3 • SC4 • Btrieve • Oracle
Informix • Clarion • DIF • SYLK
. . . plus many more file types.

- *No programming — completely menu driven!*
- *Selects & rearranges records, fields and characters!*
- *Instant script file creation for total batch processing!*
- *Maps new data structures automatically by name!*
- *Ideal for OEM/run-time applications!*

". . . an invaluable integration tool." —*Byte*
". . . more than worth its price." —*PC Magazine*
". . . a surefire winner." —*Computer Language*

Standard $99 — dBASE, ASCII, 1-2-3, Q&A, Enable, DIF, askSam & merge files for WordPerfect, Word & Wordstar
Professional $199 — Standard formats plus Binary/EBCDIC, SC4, SYLK Paradox, R:BASE, Clarion, Excel, c-tree & Btrieve
Advanced $299 — Standard & Professional formats plus Informix, Xdb, Oracle & C-ISAM

Call **800/444-1945** or 512/482-0824
MC/VISA • AMEX • COD • Corp. PO.
available for LANs, Xenix & Unix
Tools & Techniques Inc • 1620 W 12 • Austin TX 78703

T&T TOOLS & TECHNIQUES

Number **327** on the Reader Service Card

WINDOWING SCREEN MANAGER

Finally, the windowing screen manager you've been waiting for!

Tired of the frustration of forcing square pegs into round holes? So are we! That's why we created the SWS screen manager, the flexible manager that works for you. It's easy to use! It's ideal for large scale team development of user interactive applications. It's NOT another resource pig! With SWS you can prototype a complete system in minutes. You can even have numerous virtual screens open in multiple windows, but only if you want them. SWS doesn't force anything.

- Screen Management by Example
- Structured Windowing Language
- System Prototyping Tools
- Screen Porting Facilities
- Network & Mouse Support
- Builds Menu Shells for DOS
- Graphics Library for "C"
- Sight Licensing and Demo Available
- Supports: DOS, C, Pascal

Inspired Innovations
4001 South Decatur Boulevard, Suite 332
Las Vegas, NV 89103 1-800-752-8874
Professional Development Version 2.0 $320.00

Number **358** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Software Development '90

By William Roetzheim

I've just returned from the Software Development '90 conference, held in Oakland, California, from February 6 through 9. The event was held in the Oakland Convention Center in beautiful downtown Oakland, which is actually one of the least beautiful areas I've ever been in. Although the outside of the convention center looks exactly like a high-security, metropolitan correction institute, the inside contains a comfortable exhibit hall, lots of meeting rooms, and the Hyatt Hotel.

The convention itself proved to be one of those rare events where the talks were actually useful for *real* problems and *real* jobs. (The only other example I can think of is the Pacific Northwest Software Quality Conference.) In addition, it was fun meeting and talking to a hotelful of people (more than 2,500) with a common interest.

The week started with tutorial sessions scheduled for Monday and Tuesday. Ironically, the OS/2 tutorial was delayed indefinitely. (It seems *everything* about OS/2 must be delayed at least once! No one seemed to mind, though—I guess they're used to it.)

I attended the C++ tutorial taught by Bjarne Stroustrup, the originator of the language. C++ takes ordinary C, adds all of the ANSI C extensions (prototypes, and so on), and then adds data abstraction and objects. The thing that impressed me most about C++ was Bjarne's emphasis on performance. C++ code is fast and efficient (faster than C), allowing work on time-critical applications—and what application isn't time critical? I'm starting to believe something I read on a bulletin board once: "C++ is everything that Ada was supposed to be."

A quick scan of the workshop titles tells you where the industry is heading.

William Roetzheim is the author of Structured Project Manager's Toolbox (SPMT), an IBM PC–based project-management tool tailored to software development. Mr. Roetzheim may be contacted at:13518 Jamul Drive, Jamul, CA 92035 619-669-6970

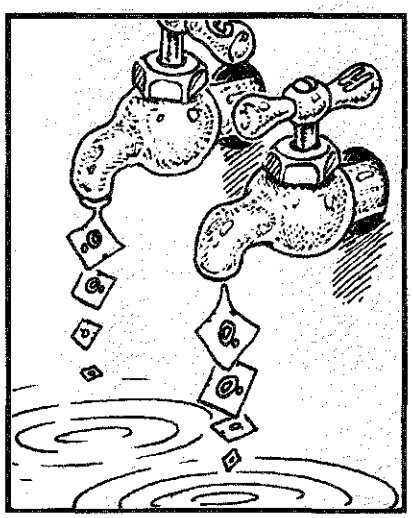


ILLUSTRATION BY JIM CARPENTER

The most noteworthy point is that if you don't know C now, you'd better learn it before your next job interview. Of the workshop titles that mentioned specific languages, C and C++ were mentioned a total of thirty-two times. The closest competitor was LISP (with two mentions), followed by Ada, BASIC, COBOL, Pascal, and Prolog (with one mention each)!

Also obvious from the list of workshop titles is that object-oriented programming in general (and, I believe, C++ in particular) is the wave of the future. Thirty-nine workshops mentioned either C++ or "object-oriented" in their title. (No one mentioned Smalltalk).

How about the operating system wars? Few people seemed to really believe that DOS would win in the long run. Four workshop titles mentioned UNIX, while twelve mentioned OS/2. Don't take this too seriously, though—the OS/2 workshops were very poorly attended. I also hear that OS/2 books are not selling at all.

At one of the workshops, Chip Rabinowitz gave an interesting talk on publishing software as shareware. Chip presented some statistics supporting the contention that successful shareware must:

- save the user time or money (which knocks out most game programs),
- appeal to a mass market (or else to programmers), and
- sell for between $25 (optimum) and $200 (risky).

At the same workshop, potential authors of shareware were referred to the Association of Shareware Producers (or ASP, P.O. Box 5786, Bellevue, WA 98006), both to join and to receive a list of ASP-approved disk distributors. (Disk distributors are those guys that advertise $1.95 a disk for software.) They are also working on a list of ASP-approved bulletin boards, which sounds like a good idea.

Space at my talk on project management ended up being standing room only. Project management seems to be the hot topic of the day, based on my recent speaking engagements. A group of us went down to the Elographics booth afterward. Elographics (405 W. Washington St., #20, San Diego, CA 92103) markets an add-on product that turns your display into a touch-screen monitor. We loaded SPMT (a project-management program I wrote) onto the computer and then had fun dragging Gantt lines around with our fingers, touching icons to select menu choices, and so on. The Elographics system seemed to be very well suited to "computer-illiterate" operators (factory floor, point-of-sale, information booths, and such).

Borland threw a nice party Wednesday evening. The Turbo Jazz Band, led by Philippe Kahn, sounded great. They were having fun in their slacks and Hawaiian shirts, and it really showed. (Somehow I think a Microsoft band would be required to wear blue suits and conservative ties—and probably play waltzes.)

I think I'll take the plunge into C++ when Borland releases their version of the language. It should be available soon—Kahn slipped at least twice during the conference to admit Borland had a working C++ compiler in-house. The Whitewater Group (the originator of Actor) mounted the stage

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

in classic Blues Brothers attire—sunglasses and all—and sang some respectable Chicago-style blues. A number called "The Programming Blues" brought the house down.

Dave Schmitt (Lattice) presented a workshop on porting DOS applications to OS/2. I'm certainly not ready to port my software yet (where's the market?), but I jotted down some notes to make sure that porting the code later will be easy. The key rules are:

- Work with large (or huge) memory models, and treat all pointers as indivisible objects; specifically, don't break a far pointer into a segment and an offset.
- Isolate all interrupt functions—int86(), intdos(), etc.—in individual modules that are easy to find.
- Isolate in separate modules all code that directly accesses video memory.

I guess that these rules are just good programming practice, anyway.

One piece of news that was especially important was the information that converting programs between Presentation Manager (OS/2) and Microsoft Windows is very difficult.

For me, the best workshop was definitely "A Little Bit on the Side," presented by Moses Ma. Moses is dynamic, sharp, and humorous, which is always a winning combination in a speaker. He talked about methods of making money from software you develop on your own (perhaps at night after work). The key rules seem to boil down to:

- Get a lawyer.
- Get an agent (if you can find one).
- Find a publisher.

I found it interesting to learn that publishers will often pay you in advance for software—that is, before you have completed it. If you are interested, send me a self-addressed envelope with two stamps, and I'll send you a list of software publishers.

The next Software Development conference is scheduled for this fall in Boston, although the East Coast version of the conference is expected to be quite a bit smaller (perhaps 300 people). I'm looking forward to next winter, when I plan to be at Software Development '91, which will be held in Santa Clara, California—that is, if they ask me to speak again.

Instant-C

INCREMENTAL C COMPILER

"An Utterly Addictive Speed Improvement"

B. Rubin, *The C Gazette*, Autumn 1989

Speed Microsoft™ C development. *Instant*-C is a Microsoft-compatible incremental compiler. It can rebuild 100,000 line programs in seconds. And you can still use Microsoft C's optimization.

Find bugs other compilers miss. *Instant*-C debugs as fast as it compiles. Here are some bugs that (in a Microsoft C program) could hang your computer. *Instant*-C catches them automatically:

- Array reference out of bounds
- Use of uninitialized pointer
- Read/write through null pointer
- Stack overflow
- Clobbering the program or DOS

For those few bugs that still elude detection, *Instant*-C integrates the debugger with the compiler for true source level debugging.

No slowdown on big programs. *Instant*-C uses our DOS/16M technology, so you can work on programs bigger than 640K with no swapping to disk.

Everything you need. *Instant*-C has much more to offer, like dynamic cross referencing and lint-like static checking. Support for third party libraries. Interactive testing of single functions or partial programs. Everything you need to program and debug faster.

Call 508-653-6006 for *free* information.

Rational Systems, Inc. 220 N. Main St., Natick, MA 01760 PJH1

Instant-C requires an 80286/386/486 and at least 1M of memory. Microsoft is a registered trademark of Microsoft Corp.

Number **20** on the Reader Service Card

Parallel Programming for "C"

INTERWORK™

A Concurrent Programming Toolkit

Interwork is a "C" program library which allows you to write your programs as a set of cooperating concurrent tasks. Very useful for real-time applications, simulation, and parallel programming.

New version 3.0 now available

FEATURES

- Unlimited number of tasks
- Fast context switching

NEW! • Optional time sliced scheduling

- Full inter-task communication

NEW! • Enhanced interrupt and trap handling

NEW! • Expanded user's guide with examples

Interwork is available for the following systems:

IBM PC, XT, AT	PC-DOS 2.0 or later	$189
IBM PC AT	XENIX 286	$239
Intel 386	UNIX V.3/386	$329
DEC VAX™, SUN 3,4	UNIX 4.2BSD	$329

For more information please contact:

Block Island Technologies

Innovative Computer Software

15455 NW Greenbrier Parkway, Suite 210
Beaverton, Oregon 97006
(503) 690-7181 FAX (503) 645-7732

Number **224** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

CC-Rider

The C Programmer's Companion
Now Available for OS/2 !!!

CCSYM - C Symbol Compiler

* Analyzes ANSI C source code. Microsoft/Turbo extensions supported.
* Automatically generates separate commented function prototype files for all global and local (static) functions.
* Log File lists all your symbol definitions with detailed usage statistics.
* Generates a "symbol database" which CCRIDER uses to quickly retrieve ANY symbol definition in your application, no matter how large it is.
* Incremental compilation supports efficient database updates.

CCRIDER - Pop-up Symbol Lookup

* Pop-up utility which converts your own editor into a powerful hypertext programming environment.
* Point at a symbol in your editor, then pop-up its definition in a window.
* One keystroke "Edit" command allows you to edit the definition in your own editor, no matter which file or directory it was defined in.
* "Paste" command inserts the definition into your currently edited file.
* Supports multiple editing sessions under OS/2.

DOS Version $89
OS/2 Version $179

FREE DEMO

Western Wares
Box C Norwood, CO 81423
(303) 327-4898

Number **100** on the Reader Service Card

4c™ Hypertext Source Code Databank for Programmers

Explore source code! Navigate uncharted logic trails easily!

4c includes:

★ **A Source Code Analyzer:**
The analyzer maps and cross-references functions and variables to their respective path and file names. The cross-reference table can be displayed alphabetically and by type of variable. It allows a programmer to edit functions directly without specifying a file name.

★ **A Point-and-Shoot Zoom Key:**
Which instantly locates any function, global variable, structure definition, #define statement, or macro, etc., in any file, directory, and drive, and displays it in a new edit window.

★ **A Full Featured Built in Editor:**
Featuring multiple files, multiple windows, cut-and-paste between files, search-and-replace, auto indent, tab support, and a familiar user interface with pull-down menus, or use your own editor!

"Given a procedure's identifier, an editor should be able to bring up the full procedure interface, including parameters, so that you don't have to remember how many and in what order. Wouldn't it be nice to hit a hot key on the name of a complicated data type and have its definition pop up in a window? It may seem like a real small matter, but how many of you still flip through printouts doing things the editor should be doing?"

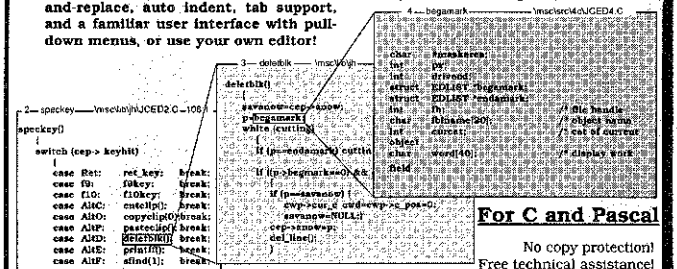
- Phillipe Kahn, "The Connection", Spring/Summer 1988

4c does this and more

"4c's Magic is apparent..."
- PC Magazine, February 1989

"Daylight in the C Confusion"
- Toolbox magazine, December 1988

"...Destined to become the standard editor for C..."
- Computer Personlich magazine, October 1988



For C and Pascal

No copy protection!
Free technical assistance!
No changes to source code required!
Required IBM PC compatible with 350K RAM.

30 Day Money Back Guarantee!
Only $119.00 plus shipping
Visa, Mastercard, check, money order, or COD.

Call or Write Today!

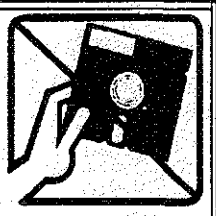
Tri-Technology Systems Inc.
1225 S. Elgin, Forest Park, Il. 60130
Phone: 312-366-7595

Number **276** on the Reader Service Card

MISOSYS

Programmer's Journal downloads are now available from MISOSYS' forum on Compuserve: GO PCS49

Are you still fussing with floppies for BACKUP? Our QIC40 JUMBO tape drive from MISOSYS is your solution!



The Colorado Memory Systems' JUMBO tape drive is one drive that fits all computers. It comes ready for internal use in AT's, XT's, and PC's, and connects to your floppy disk controller. Our adaptor board can be used to connect JUMBO when all FDC connectors are in use. Kits are available for XT/AT or PS/2 machines which convert Jumbo to external use.

- In about 5.5 minutes, JUMBO backs up 10MB's file-by-file - the fastest in the industry! 40MB's gets backed up in about 18 to 20 minutes. Uses industry-standard DC2000 tape cartridges.
- JUMBO plugs into your floppy disk controller to save cost, power, and a slot. Needs 5-1/4" (or 3.5" with faceplate) mounting slot.
- Our tape adapter board mounts in your host computer to provide an additional tape port — especially useful in 286 and 386 computers. When used with our external JUMBO, it lets you share your drive between computers. Note: external adaptor includes "Tape Adaptor"
- JUMBO has custom chips, high speed brushless motors, automatic circuits, no pots to vibrate out of calibration, and fewest parts to assure long life and technical leadership for years to come.

DJ10 Jumbo	$295 ($5S&H)
Tape Adaptor	$75 ($3S&H)
External Adaptor	$110 ($5S&H)
DC2000 tape	$22.50

M/C & VISA. S&H are U.S. only.

MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170-0239
800-MISOSYS
or 703-450-4181

Now get the clearest view of your disk drive and memory contents with our DED86™

Powerful features in Version 2!

When you need to travel through your disk drive, why settle for a tool that isolates you from the road? DED86 gives you the direct controls you need to explore your disk. It's a full-screen sector-oriented disk/file editor and a page-oriented memory editor. When you want to "unerase" erased files, DED86's flexible KEEP facility does the job without you fussing over FATs.

Check out this list of features

- ✓ Look by cylinder/head/sector, sector or cluster; even reserved/hidden sectors
- ✓ Scan free clusters to search for erased data ✓ Jump about subdirectories
- ✓ Keep sectors & clusters for writing to a file; Great for partial file recovery
- ✓ Edit bytes in hexadecimal or ASCII, zap in 0s; Undo last edit!
- ✓ Search your disk drive, a file, or memory for ASCII or hexadecimal strings
- ✓ Touch a directory file entry with your date and/or time
- ✓ Obtain complete disk statistics in one screen; includes usage data
- ✓ Alter file attributes: archive, system, hidden, read
- ✓ Save/Restore sectors to/from auxiliary buffers; for moving data around
- ✓ DOS subshell available while using DED86 lets you shell to other programs
- ✓ Handles 5.25" & 3.5" drives & RAM disks; even large partition drives

DED86 reduced by more than 50%; Now only $29.95 + $2S&H

Fujitsu Floppy Disk Drives:	5.25" 360K 1/2 height	$75+$4S&H
	3.5" 720K in 5.25" frame	$85+$3S&H
	3.5" 1.44M in 5.25" frame	$99+$3S&H

Number **121** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

386 Debugging

By John Otken

Intel's 386 CPU has built-in debugging hardware that can trap execution when the CPU reads at or writes to any of four different addresses. This article shows how the debug registers can detect stack overflows in C and assembly language.

I presume some knowledge of 386 debugging features or at least access to reference materials. If you do not have a copy, now is a good time to pick up Intel's *80386 Programmer's Reference Manual* or the i486 book.

To keep the listings as simple as possible, the code is for real-mode programs under DOS only; the code does not share the trap-interrupt or debug registers with other programs. The primary logic is in procedure **trap_memory_access** in Listing 3. To protect a word of memory, call this routine with a segment and offset in DX:AX.

The debug registers (DR0–DR3) require 32-bit linear addresses, and so the program converts the segment and offset. The address remains in EDX until two vectors are initialized over the next several lines. The first vector is the termination address in the program segment prefix, or PSP. Hooking this vector ensures that the debug breakpoints will be disabled regardless of how the program exits to DOS. The second vector is the trap interrupt (INT 1). When the data breakpoint occurs, control passes to the address at 0:4. The program stores the offset and segment of **trap_interrupt** in this vector.

The program sets the linear address in DR0 and enables breakpoint 0 in the control register DR7. The LEN0 and R/W0 bits are set to break on any data read or write access to the word at DR0. The GE bit is set to enable "exact data breakpoint matching," and the G0 bit enables breakpoint 0.

ILLUSTRATION BY JIM CARPENTER

The sample programs in C and assembly language overflow their stacks and trigger the breakpoint trap. Control transfers to **trap_interrupt**, which disables all breakpoints in control register DR7, displays the CS:IP of the instruction following the overflow, and exits with an error code of 1. In a debugging environment, **trap_interrupt** would read the debug status register, DR6, and determine the cause of the trap. This code simply assumes a stack overflow is the only possible cause.

Because the program hooks the termination vector in the PSP, control transfers to **terminate_program.** This procedure verifies that breakpoints are off and restores the original contents of interrupt vector 1. It exits to DOS by jumping to the original termination address.

STACK PROTECTION

Assembly language programs create a separate stack segment. The bottom of the stack is offset 0 of this segment. Be sure to use the STACK segment and not DGROUP. In the sample program, I trigger the stack overflow 32 bytes from the bottom so that the stack pointer won't wrap. Checking for stack overflow in C is a bit different.

Microsoft C programs call the **__chkstk** procedure upon entry to each C function. Register AX holds the number of bytes of local storage to allocate. The procedure allocates space on the stack and compares the new SP with the DGROUP variable STKHQQ. If SP is below STKHQQ, the program aborts with a stack overflow.

The sample program sets the 386 debug registers to protect the word corresponding to SS:STKHQQ. The function **overflow_stack** recurses until the debug breakpoint. This scheme is not perfect. If unused local-variable space falls at SS:STKHQQ, the debug breakpoint will not occur. I also assume Microsoft C will not change the value at STKHQQ during program execution.

To use the 386 breakpoint feature for stack protection, turn off normal stack checking with the */Gs* switch. To enable stack checking, call **protect_msc_stack** with no arguments. If your program uses the medium or large model, add "far" to the "proc" statement in Listing 2.

EXACT DATA BP MATCHING

Using the debug registers affects 386 performance. According to the 486 manual, the CPU disables instruction overlap and executes only one instruction at a time. This is necessary so that the CPU can trap on the offending instruction and not on one several instructions later.

This feature is called "exact data breakpoint matching" and the control-register bits GE or LE enable this feature. You should set one of these bits even if you don't care to know the exact address of the offending instruction. The 486 manual says, "If exact data breakpoint match is not selected, data breakpoints may not be reported until several instructions later or may not be reported at all." My 386 does not report a stack overflow when I clear those bits.

Have fun.

John Otken is the president of Soft Advances, where he develops software for embedded systems. He is a frequent contributor of assembly language articles to PJ, including the one on page 64 of this issue. John can be reached at 213-559-7015.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
        title   test 386 stack trap for assembly language
        dosseg
        .model  small
        .stack  100h

        .386p
        .code
        extrn   trap_memory_access:near

;;      main
;
main    proc
        mov     dx,STACK                ; break on access to stack
        mov     ax,32                   ;  bottom+32
        call    trap_memory_access

        mov     cx,80h                  ; overflow stack
mail:   push    cx
        loop    mail

        mov     ax,4C00h
        int     21h
main    endp

        end     main
```

LISTING 2

```
        title   test 386 stack trap for Microsoft C
        .model  small
        .data
        extrn   STKHQQ:word
        .386p
        .code
        public  _protect_msc_stack
        extrn   trap_memory_access:near

;;      _protect msc stack
;
_protect_msc_stack proc
        mov     dx,ss                   ; ss:STKHQQ is stack bottom for
        mov     ax,@data                ;  Microsoft C 5.x
        mov     es,ax
        mov     ax,STKHQQ
        call    trap_memory_access
        ret
_protect_msc_stack endp

        end
```

LISTING 3

```
        title   386 memory protect
        .model  small

        .data
original_int1   label   dword
        dw      trap_interrupt,@code

terminate_address label dword
        dw      terminate_program,@code

stack_over_text db      'Stack overflow at CS:IP==$'


        .386p
        .code


;;      outchr
;
;       entry   AL      character to display
;       uses    AX
;
outchr  proc
        push    dx
        mov     dl,al
        mov     ah,2
        int     21h
        pop     dx
        ret
outchr  endp


;;      outchr hex word
;
;       entry   AX      word
;       uses    AX
;
outchr_hex_word proc
        push    ax
        mov     al,ah
        call    ohw1
        pop     ax
ohw1:   push    ax
        sar     al,1
        sar     al,1
        sar     al,1
        sar     al,1
        call    ohw2
        pop     ax
ohw2:   and     al,0Fh
        add     al,90h
        daa
        adc     al,40h
        daa
        jmp     outchr
outchr_hex_word endp


;;      terminate program
;
terminate_program proc
        mov     eax,0                   ; disable breakpoints
        mov     dr7,eax

        mov     ax,@data                ; restore original interrupt 1
        mov     ds,ax
        mov     eax,original_int1
        mov     dx,0
        mov     es,dx
        mov     es:[4],eax

        jmp     terminate_address       ; goto original terminate address
terminate_program endp


;;      trap interrupt
;
trap_interrupt proc
        mov     eax,0                   ; disable breakpoints
        mov     dr7,eax

        mov     ah,9                    ; *Stack overflow at CS:IP==*
        mov     dx,@data
        mov     ds,dx
        lea     dx,stack_over_text
        int     21h

        pop     dx                      ; display CS:IP
        pop     ax
        call    outchr_hex_word
        mov     al,':'
        call    outchr
        mov     ax,dx
        call    outchr_hex_word

        mov     ax,4C01h                ; exit program
        int     21h
trap_interrupt endp
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;;      trap memory access
;
;       entry   DX:AX   real address to protect
;       uses    EAX,BX,EDX,ES
;
trap_memory_access proc
        push    ds
        movzx   eax,ax                  ; convert real-mode dword address
        movzx   edx,dx                  ;  to linear address
        shl     edx,4
        add     edx,eax

        mov     ah,30h                  ; check DOS version
        int     21h
        cmp     al,3
        jb      tma3                    ;  if before DOS3

        mov     ah,62h                  ; get PSP (requires DOS3)
        int     21h
        mov     es,bx

        mov     ax,@data                ; set terminate address at PSP:0Ah
        mov     ds,ax
        mov     eax,terminate_address
        cmp     ax,offset terminate_program
        jne     tma1                    ;  if terminate address set already
        xchg    eax,es:[0Ah]
        mov     terminate_address,eax

tma1:   mov     ax,0                    ; set trap interrupt vector
        mov     es,ax
        mov     eax,original_int1
        cmp     ax,offset trap_interrupt
        jne     tma2                    ;  if set already
        xchg    eax,es:[4]
        mov     original_int1,eax

tma2:   mov     dr0,edx                 ; enable global breakpoint 0 for any
        mov     eax,070202h             ;  16 bit read or write to linear
                                        ;  adr
        mov     dr7,eax

tma3:   pop     ds
        ret
trap_memory_access endp

        public  trap_memory_access,trap_interrupt,terminate_program

        end
```

LISTING 4

```
main()
{
        protect_msc_stack();
        overflow_stack();
}

overflow_stack()
{
        overflow_stack();
}
```


MORE 34010 Video Co-Processors from PC Tech

- Mono 736 x 1024 - 2048 x 1536
- Color 1024 x 768 (1280 x 1024 soon!)
- 512K - 4M DRAM
- Optional CCITT & DGIS support
- **AGE XoftWare X Window Server certified board**
- C Compiler, Assembler, Linker, Debugger, DGIS Development Kit & TIGA Toolkit available
- **TIGA** included with all boards
- Custom designs available
- **PC Tech understands the 34010**

For further information, please contact:

PC Tech

907 N. 6th St.
Lake City, MN 55041
(612) 345-4555 (Voice)
(612) 345-5514 (FAX)

PC Tech is a trademark of PC Tech, Inc.

PC, XT, AT, DGIS, XoftWare, and DGIS are trademarks or registered trademarks of their respective companies.

Number **208** on the Reader Service Card

Herc EGA VGA SVGA Unix Xenix Graphics Tool Kit

Now Supports 14 Video Modes! *including 800x600x16 and 360x480x256 on VGA systems, 640x400x256 on Paradise SVGA cards, and 800x600x256 on Orchid Designer and Micron cards.*

No CGI or BIOS or X11 - Direct Video Card Access: *Very Fast!*

$85 - Image Tool Kit: Display GIF Images in all video modes on multiple virtual terminals. Scale image to screen size or pan. Modify color map. Convert between GIF and many other formats. Cut, paste, enlarge, shrink, convert to gray scale.

$155 - Programmer's Tool Kit: Binary License to C library with point, line, solid, bitblt, clipping, ellipse, region fill and arc primitives all implemented without floating point math. Software cursor for bus mouse. Fifty BDF fonts. Twelve alu operations in 16 color modes. And one binary can run on any type of card in any video mode. Source to GIF viewer and several demonstration programs.

$299 - Complete C Source Code: Parts written in assembly. NO ROYALTIES in most circumstances, and fully documented.

BONUS: The best in Unix Public Domain graphics C source code with drivers for the Programmer's Tool Kit, including:

A device-independent (Unix/Xenix, DOS, POSTSCRIPT, X11) 2-d and 3-d plot library featuring 3-d transformations (including fonts) and various types of curve fits.
A collection of ray tracers.

386/ix (all versions) - SCO Xenix 2.3 - AT&T System V/386 - Esix - SCO Unix SPECIFY VERSION WHEN ORDERING

Send for More Information and a Demo!

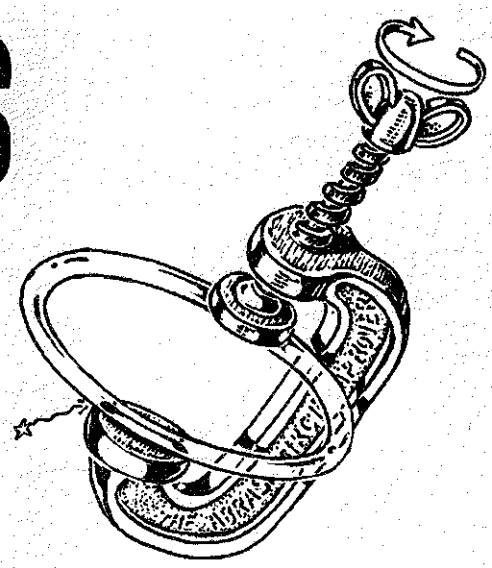
JDyx *Enterprises*
907 Tuxworth Circle, Decatur, GA 30033
(404) 320-7624 BBS: (404) 321-5020

Number **346** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

By Michael Abrash

What goes around



Drawing ellipses on the VGA

First things first. The best circle-drawing routine I know of gets faster seemingly by the day—so much so that I'm beginning to think that the ultimate circle-drawing routine will consist of just a single **nop** once we figure out how to cut all the superfluous instructions. Consider this:

Hal Hardenbergh's circle-drawing approach (presented over the last two installments) drew circles faster than I would have thought was possible. Even so, in the last installment I went to great pains to point out that I did *not* think that my implementation of Hal's approach was the fastest possible way to draw circles but just one good way among many.

Good thing, too, because just after that issue went to press, I chanced to talk once again with Hal. I mentioned that there was really no need for even the few multiplications used in his algorithm: The squared terms canceled out right at the beginning, allowing circles to be drawn without a single multiply or divide. Hal thought about that for a while and realized that, given the elimination of the squared terms, we didn't need 32-bit integers any more; plain old 16-bit integers would do just fine. Whereupon he devised yet another circle-drawing algorithm using 16-bit integers and requiring less than ten instructions per point to generate a circle arc.

I suspect that Hal's new algorithm can be used to draw circles faster than the Bresenham's line-drawing algorithm I presented back in installment 12 [*PJ* 7.3 (May/June 1989)]. I'm not going to present Hal's new circle-drawing algorithm here, for a couple of reasons: I've probably overdosed you on circles by this point, and Hal may want to publish his new findings himself. If you know what you're doing, it should be easy to apply the above information to my discussions in the last two installments to derive the new approach. Of course, if I get enough requests, I'll talk with Hal and probably run an installment called "*Really* Fast Circles for the VGA (We're Not Kidding this Time!)" or some such; I'm always happy to let the readers set the agenda.

At any rate, if you think your code is optimized to the limit, *maybe* it is—but don't bet your house on it. And with that in mind, let's get on with learning how to draw ellipses pretty doggone fast.

An ellipse is an oval, which is to say a squashed (or, possibly, nonsquashed) circle. Ellipses are centered around two foci; the sum of the distances from the two foci to any point on a given ellipse is a constant. A circle is actually a special case of ellipse for which every point is equidistant from the center, where both foci reside one atop the other.

UNDERSTANDING ELLIPSES

A circle has a single radius that is the distance from the center to every point on the circle. An ellipse has two basic radii, one that is the distance from the center of the ellipse to the edge along the x axis and one that is the distance along the y axis, as shown in Figure 1. These two distances, which we'll call the x radius (of length a) and the y radius (of length b), along with the center of the ellipse (the point where the x and y radii meet) are the fundamental parameters we'll work with; we won't concern ourselves with the foci or distances from the foci from now on. You might think of ellipses as being defined by the smallest rectangle that contains them; however, for our purposes, there's an additional limitation that A and B must be integers, and so the encompassing rectangle must have even dimensions to allow the center to fall squarely on a pixel. (It's possible to support ellipses with fractionally located centers; since it's generally unnecessary and it complicates matters, we'll avoid it.)

CALCULATING CIRCLES

Circles are certainly faster and easier to calculate than ellipses. Where the equation for a circle is

$$x^2 + y^2 = r^2$$

Contributing Editor Michael Abrash is a senior engineer working for Metagraphics, of Scotts Valley, California. He is the author of Power Graphics Programming *and* Zen of Assembly Language. *Readers may write to Michael at P.O. Box 414, Burlington, VT 05402-0414, or contact him via MCI Mail, user name* MABRASH.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The capability to draw ellipses is . . . essential for drawing circles on most PC displays.

the equation for an ellipse is

$x^2/a^2 + y^2/b^2 = 1$

(Make a and b the same and you get a circle.)

By the way, the equation above is specifically for nontilted ellipses—ellipses with horizontal and vertical axes, where the foci share either a common x coordinate or a common y coordinate—and that's all we'll work with for now. We'll get to tilted ellipses by and by; they're flexible and useful, but they're also slower, more complicated, and generally quite a different kettle of fish from nontilted ellipses.

WHY ELLIPSES MATTER

The question, as always, is how to get a PC to draw the object implied by the above equation for a nontilted ellipse as quickly as possible, and that's what we'll spend this installment and the next figuring out. It's well worth knowing how to draw ellipses fast, for they're most useful. Of course, it's often necessary to draw ovals, and ellipses are handy for that alone, but there's more to it than that.

You see, true circles, of the sort we learned to draw in the last installment, are useless on displays with nonsquare pixels (displays with aspect ratios other than 1:1): On such displays, true circles, which space pixels evenly in both directions, appear to be ellipses. Examples of such displays are Hercules graphics, all EGA graphics modes, and all standard VGA graphics modes except 640x480. In other words, you can't use a true circle-drawing algorithm to draw circles in any standard IBM mode except one.

You *can* use ellipses to draw circles in all those modes, though: Just adjust the ratio of the x and y radii of each ellipse to balance the aspect ratio of the display, and bingo! you have a circle. For example, the aspect ratio of the EGA mode 10h display is about 4:3 (four pixels in the x direction covers the same distance as three pixels in the y direction); so if you draw an ellipse with an eccentricity (a/b ratio) of 4:3, it will appear as a circle in mode 10h.

We'll see an example of this shortly. The important point is that the capability to draw ellipses is not only useful for drawing ovals but essential for drawing circles on most PC displays.

DIVIDE AND CONQUER

Now that we've established that ellipses are good stuff, how will we draw them fast? The path we'll take will be familiar to those of you who followed the past two installments on circles. First, we'll learn how to draw ellipses using the basic ellipse equation and floating-point arithmetic. Next, we'll derive a far more efficient algorithm—one that uses no noninteger arithmetic or square roots and that requires no multiplies or divides in the main loops—and implement it in C. That will take us to the end of this installment. In the next installment, we'll tune the C implementation to the quirks of the EGA and VGA, and finally we'll convert the critical code to assembly language.

I hadn't planned on ellipse drawing being a two-parter, but there's sure as heck no way I can squeeze it into one installment. In fact, I'm tight for space even in two installments, which leads to some terser, more cryptic explanations—ironically, precisely the symptoms I railed against when I started to write about circle drawing. My apologies for that, but I'm doing the best I can given space constraints; anyway, most readers just want fast working code and don't care about how the implementation works, and fast working code they'll get. I actually considered skipping the discussion of how ellipse drawing works altogether to save space, but decided against that for one simple reason: If you don't understand how my code works, you can't make it better—and part of the pleasure of this column is having the readers take my code and shoot it back to me much improved. (Just look at what Hal did with my circle-drawing implementation.)

In short, if my explanations are less complete than you might wish, I'm sorry about that, but all the critical knowledge is in there. Ferret it out and put it to work, and when you're done, please share with *PJ*'s readers how ellipses can be done better.

That said, let's learn how to draw ellipses. We'll start by drawing them slowly, but you can be sure that *that* will change.

THE EASY WAY

The straightforward way to draw an ellipse is implemented in Listing 1. This listing takes advantage of the four-way symmetry of nontilted ellipses, shown in Figure 2, by generating one arc in which x is the major axis (that is, the x coordinate advances faster) and drawing all four symmetries at once via four calls to a dot-plot function, and then generating one arc in which y is the major axis and drawing all four symmetries the same way.

Arc generation is accomplished by starting where the major axis is 0 relative to the center of the ellipse and the minor axis is the maximum distance from the center and then stepping the major axis by one pixel each time and calculating the corresponding minor-axis point via floating-point arithmetic according to the ellipse equation above. This continues until the arc reaches the point at which the major axis changes, which is the point at which the slope of the arc reaches 45 degrees.

So, for example, when drawing an arc for which x is the major axis, the initial point drawn would be $(0,b)$. The next point drawn would be $(1,y)$, where y is calculated as

$x^2/a^2 + y^2/b^2 = 1$

$y^2/b^2 = 1 - x^2/a^2$

$y^2 = b^2 - b^2 * x^2/a^2$

so

$y = \text{sqrt}(b^2 - b^2 * x^2/a^2)$

rounded to the nearest integer. The same calculation is repeated for each x as x is incremented along the arc, with the calculated coordinates reflected around the ellipse. (All x and y coordinates discussed are relative to the center of the

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ellipse.) That's easy enough, eh? The only trick is knowing when to stop, and that happens when the *y* component of

$x^2/a^2 + y^2/b^2 = 1$

is no longer the larger component, as detected by

$y^2/b^2 <= x^2/a^2$

As you'd expect, the same thing is done for the arc where *y* advances faster, but with *x* swapped with *y* and *a* swapped with *b* in the calculations.

Link Listing 1 to Listing 2 to see ellipses drawn to appear as circles in mode 10h. Some multiscanning monitors don't provide a 4:3 aspect ratio in mode 10h, and so the ellipses may not look all that circular on those monitors. Trust me, they do have an eccentricity of 1.33. Link Listing 1 to Listing 3 to see ellipses drawn with a variety of eccentricities.

When you run these listings, you will see that ellipses drawn by stepping the major axis tend to look rather jagged; that's the cost of performance, and their appearance is certainly acceptable. Perhaps someday I'll discuss anti-aliasing, which in 256-color modes can produce stunning results—but I promise you that anti-aliased ellipses would be orders of magnitude slower than the step-drawn ones we'll see shortly.

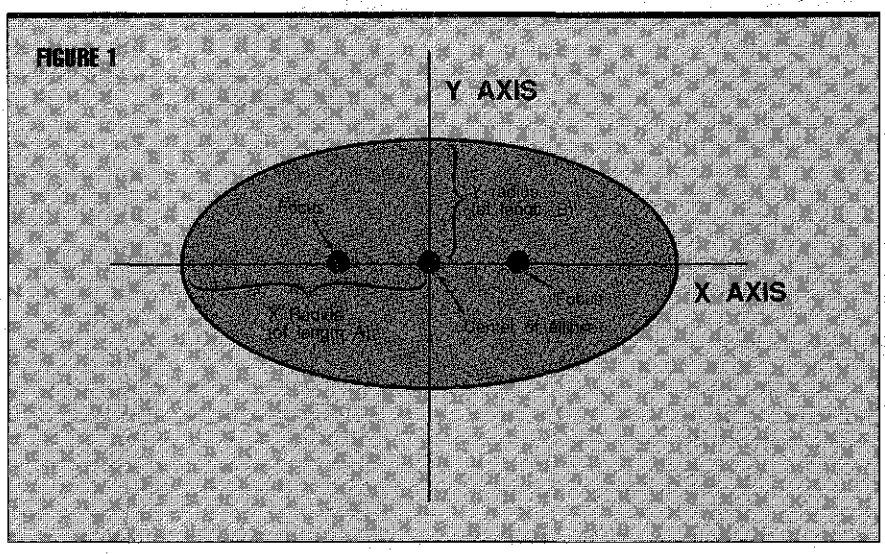
As you can see, drawing an ellipse is no great trick; however, drawing an ellipse fast requires a little more thought.

AN INCREMENTAL APPROACH

Remember the incremental, integer-only algorithm we used to speed up circle drawing two installments back? I hope you do, because fast ellipse drawing, as implemented in Listing 4, is strikingly similar, if slightly more complicated, and I'm not going to discuss the process in so much detail this time. Basically, instead of calculating the minor-axis coordinate from scratch for each pixel, the incremental approach calculates it as a delta from the last pixel.

The tremendous advantage of the incremental approach is that all terms used can be maintained as integers rather than floating-point values. Better yet, no multiplication or division is required to advance from one point to the next (not counting multiplication by 2, which is really an add or shift). The incremental approach isn't quite so fast for ellipses as for circles, but it nonetheless makes ellipses *nearly* as fast as the circles we've drawn in the last two installments, and that's remarkably fast.

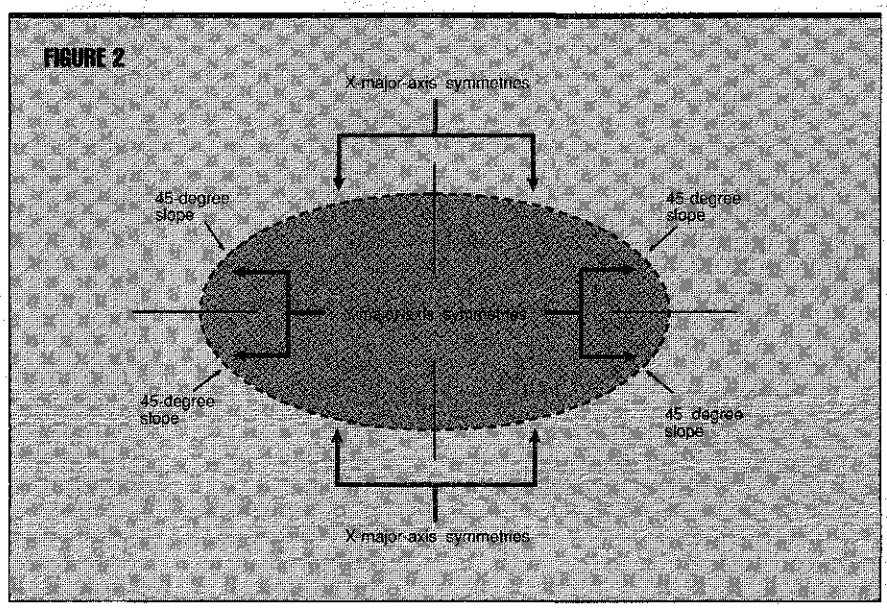
That said, let's take a quick trip through the math of the incremental ellipse-drawing approach for those of you with a mind to do some tinkering on your own.



A THUMBNAIL DERIVATION

As with the floating-point approach, the incremental approach draws ellipses by generating an arc for which *x* advances more rapidly than *y*, drawing the four symmetries, and then doing the same for *y*. Also like the floating-point approach, the coordinate that advances more rapidly—the major axis for the arc being drawn—is incremented by one each time, and the corresponding minor-axis point is drawn. The only difference between the two approaches lies in the way the minor-axis coordinate is determined. Where the floating-point approach recalculates the minor-axis coordinate, the incremental approach merely decides whether the minor-axis coordinate has changed from the previous point and advances that coordinate by one if that is the case.

The trick to the incremental approach, then, lies in deciding when it's time to advance the minor-axis coor-



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HI-SCREEN XL™

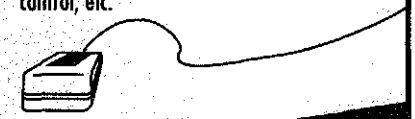
THE PROVEN USER-INTERFACE SOLUTION FOR YOUR PROGRAMS

Since 1983, Softway has specialized in user-interface development tools aimed at demanding professional programmers.

WINDOWS · MENUS DATA ENTRY · HELP

HI-SCREEN XL offers a high-level modular approach to screen management that provides for superior programming **control, flexibility** and **speed.**

- The built-in **screen editor** is used to create screens/windows/menus along with all field descriptions. Any **Hercules, CGA, EGA** or **VGA** graphics can also be captured. Screens are then saved in a compressed file format outside your program.
- **High-level calls** from your program give you total control over the user interface. Combine text and graphics, dialog boxes, context-sensitive help, elaborate data entry & menu management, keyboard & mouse control, etc.



WITH MOUSE AND GRAPHICS SUPPORT

- Supports all versions of **C, Basic, Cobol, Fortran, Pascal, dBase, Assembly,** etc.
- Use linkable (with C and QuickBasic) or removable TSR screen driver (30-40K).
- 2 compatible versions:

HI-SCREEN XL $149
HI-SCREEN XL *Professional Series*
(with mouse & graphics support): $325

30 day money-back guarantee • No royalties

Call today: **1-800-338-2852**
in CA: 415-397-4666 • Fax: 415-397-4677

SOFTWAY, Inc.
500 Sutter St., Suite 222, San Francisco, CA 94102

Number 110 on the Reader Service Card

dinate. Here, in a nutshell, is how that works.

The equation for an ellipse is

$$x^2/a^2 + y^2/b^2 = 1$$

which can be expressed as

$$b^2 * x^2 + a^2 * y^2 - a^2 * b^2 = 0$$

When drawing an arc for which x is the major axis, all we'll do is evaluate the above equation initially for $x = 0$ and $y = b - 0.5$, which will give us a measure of how far off x is from the value at which y does equal $b - 0.5$ ($b - 0.5$ being the point at which y gets closer to the next pixel and advances). Then we'll reevaluate the equation for $x + 1$ each time x advances one pixel.

When the result becomes positive, we'll have passed the point at which y is closer to the next pixel; so, we'll decrement y and adjust the equation for the new y value, and then start looking in the same way for the next time y advances. That's really all there is to it.

So, when drawing an arc for which x advances faster, we'll set the initial y to $b - 0.5$ and the initial x to 0, which when plugged into the above equation gives us

$$b^2 * 0^2 + a^2 * (b - 0.5)^2 - a^2 * b^2 = 0$$

which is to say (squaring $b - 0.5$)

$$0.25 * a^2 - a^2 * b = 0$$

So the initial threshold is $a^2/4 - a^2 * b$. This is easy enough to calculate with integer arithmetic. True, we're ignoring a possible fractional term from $a^2/4$, but that's no problem; we'll simply choose to advance y if the threshold becomes exactly 0, thereby correctly handling the case where there's an implied fractional value.

MAKING ADJUSTMENTS

Now that we have an initial threshold, we have to adjust it each time we advance x until it becomes positive, indicating a change in y. That's done by advancing the x-based component of the threshold from

$$b^2 * x^2$$

to

$$b^2 * (x + 1)^2$$

which can be expressed as

$$b^2 * (x^2 + 2 * x + 1)$$

or

$$b^2 * x^2 + b^2 * 2 * x + b^2$$

Since $b^2 * x^2$ is already in the current threshold equation,

$$b^2 * 2 * x + b^2$$

is added to the x-based term each time x advances, and that quantity can be maintained with integer arithmetic on an ongoing basis.

When the threshold is reached or exceeded, the y coordinate is decremented by one, and the threshold must be adjusted back down in preparation for the next advance. This is accomplished by adjusting the y component of the ellipse equation to

$$a^2 * (y - 1)^2$$

which is

$$a^2 * (y^2 - 2 * y + 1)$$

or

$$a2 * y^2 - a^2 * 2 * y + a^2$$

Since $a^2 * y^2$ is the value we're adjusting from, the incremental portion of this equation is simply $-a^2 * 2 * y + a^2$, and, like the x component above, that value is easily maintained with integer arithmetic as y advances. (Note, however, that the y in the above equation has a fractional component of 0.5 that ends up canceling the a^2 added at the end of the equation. See installment 16 [*PJ* 8.1 (January/February 1990)] for a more detailed discussion of this phenomenon in the context of circles.)

Drawing stops when the 45-degree point is reached. That condition is detected when the minor-axis adjustment equals or exceeds the major-axis adjustment, thereby becoming the dominant component in calculating the threshold and causing the arc to advance more rapidly along the minor axis.

Quick and sketchy as this explanation is, everything you really need to understand how the incremental approach works for ellipses is laid out above; if you had trouble following this, you might refer back to the lengthier explanation of the incremental approach for circles in installment 16.

. . . Or you might not. After all, what you really need to know is how to draw ellipses fast, and Listing 4 does that, with better yet to come in the next installment.

NOTES AND CAVEATS

Unlike our circle-drawing code, the ellipse-drawing code in Listing 4 won't handle radii as large as 32K; the limit varies depending on the radii combination, but it is never less than 1K. Given that 511 is the largest usable nonclipped

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

radius on the highest-resolution Super-VGA available, a limit of 1K shouldn't pose any problem. If a greater range is needed, integers larger than 32 bits could be used, although that's more easily done in assembly language than in C. Along the same lines, calculations for smaller ellipses could potentially be performed using smaller integers. In general, I've made no particular attempt to optimize the code presented in this installment. Next time we'll worry about fine-tuning, which is pointless without the foundation of a good algorithm such as the one we've developed.

The incremental approach used in Listing 4 is not Hal Hardenbergh's ellipse-drawing approach. Hal has come up with a fixed-point technique that is slightly less precise owing to fractional round-off but looks faster than the approach I've presented. Notwithstanding that this is not Hal's approach, he patiently let me bounce it off him and kept me from getting wildly off track, for which I am most grateful.

I derived the approach I've presented from the circle-drawing approach we've already covered because I prefer exact plotting when I can get it at little cost and because I thought it would be easier for readers to understand an extension of what we've already covered than something new. Perhaps Hal will publish his approach; if you're really interested, I imagine he'd send you a copy if you wrote him via *PJ* and asked politely.

HOW FAST IS IT?

The big question, of course, is: How much faster is the incremental approach? *Plenty.* Listing 2 runs about twenty times faster when linked to the version of **DrawEllipse** in Listing 4 than to the version in Listing 1, and Listing 3 produces similar results. A numeric coprocessor would help Listing 1, but not *that* much; anyway, you can't count on every system having a coprocessor.

And we've only begun to kick ellipse drawing into high gear.

IN CONCLUSION

And so we arrive halfway through our exploration of drawing nontilted ellipses, having succeeded in developing a faster algorithm. In the next installment, we'll tackle the two remaining legs of the optimization sequence by tailoring the code to the EGA/VGA and converting to assembler. Based on our experience with circle drawing, I'd expect a two- to three-times additional performance improvement, with the tally for our final code running at around forty to sixty times faster than Listing 1.

After the next installment, we may take a break from drawing round objects before plunging back into tilted ellipses. I've received a request to discuss the VGA's color-generation capabilities, which sounds good to me; other suggestions are welcome.

See you next time.

LISTING 1

```
/* Draws an ellipse of the specified X and Y axis radii and color,
 * using floating-point calculations.
 * Compiles with either Turbo C 2.0 or MSC 5.0.
 * VGA or EGA.
 */

#include <math.h>
#include <dos.h>

/* Turbo C accepts outp for outportb, but not outpw for outport */
#ifdef __TURBOC__
#define outpw  outport
#endif

#define SCREEN_WIDTH_IN_BYTES 80    /* # of bytes across one scan
                                       line in modes 10h and 12h */
#define SCREEN_SEGMENT        0xA000 /* mode 10h/12h display memory seg */
#define GC_INDEX              0x3CE /* Graphics Controller Index port */
#define SET_RESET_INDEX       0     /* Set/Reset reg index in GC */
#define SET_RESET_ENABLE_INDEX 1    /* Set/Reset Enable reg index in GC */
#define BIT_MASK_INDEX        8     /* Bit Mask reg index in GC */

/* Draws a pixel at screen coordinate (X,Y) */
void DrawDot(int X, int Y) {
   unsigned char far *ScreenPtr;

   /* Point to the byte the pixel is in */
#ifdef __TURBOC__
   ScreenPtr = MK_FP(SCREEN_SEGMENT, (Y*SCREEN_WIDTH_IN_BYTES) + (X/8));
#else
   FP_SEG(ScreenPtr) = SCREEN_SEGMENT;
   FP_OFF(ScreenPtr) = (Y * SCREEN_WIDTH_IN_BYTES) + (X / 8);
#endif

   /* Set the bit mask within the byte for the pixel */
   outp(GC_INDEX + 1, 0x80 >> (X & 0x07));

   /* Draw the pixel. ORed to force read/write to load latches.
      Data written doesn't matter, because set/reset is enabled
      for all planes. Note: don't OR with 0; MSC optimizes that
      statement to no operation */
   *ScreenPtr |= 0xFF;
}

/* Draws an ellipse of X axis radius A and Y axis radius B in
 * color Color centered at screen coordinate (X,Y). Radii must
 * both be nonzero */
void DrawEllipse(int X, int Y, int A, int B, int Color) {
   int WorkingX, WorkingY;
   double ASquared = (double) A * A;
   double BSquared = (double) B * B;
   double Temp;

   /* Set drawing color via set/reset */
   outpw(GC_INDEX, (0x0F << 8) | SET_RESET_ENABLE_INDEX);
                              /* enable set/reset for all planes */
   outpw(GC_INDEX, (Color << 8) | SET_RESET_INDEX);
                               /* set set/reset (drawing) color */
   outp(GC_INDEX, BIT_MASK_INDEX); /* leave the GC Index reg pointing
                                      to the Bit Mask reg */

   /* Draw the four symmetric arcs for which X advances faster (that is,
      for which X is the major axis) */
   /* Draw the initial top & bottom points */
   DrawDot(X, Y+B);
   DrawDot(X, Y-B);

   /* Draw the four arcs */
   for (WorkingX = 0; ; ) {
      /* Advance one pixel along the X axis */
      WorkingX++;

      /* Calculate the corresponding point along the Y axis. Guard
         against floating-point roundoff making the intermediate term
         less than 0 */
      Temp = BSquared - (BSquared *
         (double)WorkingX * (double)WorkingX / ASquared);
      if ( Temp >= 0 ) {
         WorkingY = sqrt(Temp) + 0.5;
      } else {
         WorkingY = 0;
      }
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
    if (((double)WorkingY/BSquared) <= ((double)WorkingX/ASquared))
       break;

    /* Draw the 4 symmetries of the current point */
    DrawDot(X+WorkingX, Y-WorkingY);
    DrawDot(X-WorkingX, Y-WorkingY);
    DrawDot(X+WorkingX, Y+WorkingY);
    DrawDot(X-WorkingX, Y+WorkingY);
 }

 /* Draw the four symmetric arcs for which Y advances faster (that is,
    for which Y is the major axis) */
 /* Draw the initial left & right points */
 DrawDot(X+A, Y);
 DrawDot(X-A, Y);

 /* Draw the four arcs */
 for (WorkingY = 0; ; ) {
    /* Advance one pixel along the Y axis */
    WorkingY++;

    /* Calculate the corresponding point along the X axis. Guard
       against floating-point roundoff making the intermediate term
       less than 0 */
    Temp = ASquared - (ASquared *
       (double)WorkingY * (double)WorkingY / BSquared);
    if ( Temp >= 0 ) {
       WorkingX = sqrt(Temp) + 0.5;
    } else {
       WorkingX = 0;           /* floating-point roundoff */
    }

    /* Stop if Y is no longer the major axis (the arc has passed the
       45-degree point) */
    if (((double)WorkingX/ASquared) < ((double)WorkingY/BSquared))
       break;

    /* Draw the 4 symmetries of the current point */
    DrawDot(X+WorkingX, Y-WorkingY);
    DrawDot(X-WorkingX, Y-WorkingY);
    DrawDot(X+WorkingX, Y+WorkingY);
    DrawDot(X-WorkingX, Y+WorkingY);
 }

 /* Reset the Bit Mask register to normal */
 outp(GC_INDEX + 1, 0xFF);

 /* Turn off set/reset enable */
 outpw(GC_INDEX, (0x00 << 8) | SET_RESET_ENABLE_INDEX);
}
```

LISTING 2

```
/* Draws a series of concentric ellipses that should appear to be
 * circles in the EGA's hi-res mode, mode 10h. (They may not appear
 * to be circles on monitors that don't display mode 10h with the
 * same aspect ratio as the Enhanced Color Display.)
 * For EGA or VGA.
 * Compile and link with listingX.c (where X is 1 or 4) with:
 *    tcc listingX listing2      (Turbo C)
 *    cl listingX.c listing2.c   (MSC)
 */

#include <dos.h>

main() {
   int BaseRadius, Temp, Color;
   union REGS Regs;

   /* Select EGA's hi-res 640x350 graphics mode, mode 10h */
   Regs.x.ax = 0x0010;
   int86(0x10, &Regs, &Regs);

   /* Draw concentric ellipses */
   for ( BaseRadius = 2, Color = 7; BaseRadius < 58; BaseRadius++ ) {
      DrawEllipse(640/2, 350/2, BaseRadius*4, BaseRadius*3, Color);
      Color = (Color + 1) & 0x0F;   /* cycle through 16 colors */
```


ProGraphx

ProGraphx Toolbox™

Over **50** powerful hi-speed graphics routines for EGA/VGA graphics adapters, including proprietary icon routines to display color images.

ProGraphx Image Editor, a complete color paint package to design color icon images. Edit and view up to 50 icons at once. Simply paint an icon, then use the ProGraphx Toolbox™ routines to display it with amazing speed. Over **500** sample icons for use in your own programs.

Operation:Airlift, a full screen scrolling arcade style game with source code to demonstrate the power and speed of the ProGraphx Toolbox™.

All routines written in assembly for fastest possible performance.

Available for Turbo Pascal, or C.

ProGraphx Menus™

Menus and icons without the work!

Mouse driven pull-down menu system. Use a simple text file format to define menu structure along with event codes, then compile and use in application. Only 4 lines of code to add. Event driven menus can be changed and rearranged external to program without recompiling application. ***ProGraphx Menus***™ uses the fast routines in ***ProGraphx Toolbox***™ to create the fastest response possible. **Powerful, yet easy to implement!**

Also, function-key keyboard menus. Implemented in both text and graphics mode for conformity of applications. Simple to use.

Complete source code for menu-based microcoded processor simulator, along with memory and microcode editor. Provides examples of all windowing routines, in both text and graphics mode.

Available for Turbo Pascal, or C.

ProGraphx Runtime/Source

ProGraphx ToolBox™	$199
Toolbox with Source	399
ProGraphx Menus™	99
Menus with Source	199
ProGraphx Demo Disk	9.95

*Specify Compiler and Version.
**Illinois Residents add 8% sales tax.

ProGraphx
Speed, Simplicity, and Power for Graphics Programmers

Send Check or Money Order to:
Stanwood Associates, Inc.
303 East Ohio Street
Chicago, IL 60611

With VISA/MC Call
1-(800) 752-7982
(312) 828-9734
FAX: (312) 828-9742

Number **330** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
    }

    /* Wait for a key, restore text mode, and done */
    scanf("%c", &Temp);
    Regs.x.ax = 0x0003;
    int86(0x10, &Regs, &Regs);
}
```

LISTING 3

```
/* Draws nested ellipses of varying eccentricities in the VGA's
 * hi-res mode, mode 12h.
 * For VGA only.
 * Compile and link with listingX.c (where X is 1 or 4) with:
 *    tcc listingX listing3      (Turbo C)
 *    cl listingX.c listing3.c   (MSC)
 */

#include <dos.h>

main() {
    int XRadius, YRadius, Temp, Color;
    union REGS Regs;

    /* Select VGA's hi-res 640x480 graphics mode, mode 12h */
    Regs.x.ax = 0x0012;
    int86(0x10, &Regs, &Regs);

    /* Draw nested ellipses */
    for ( XRadius = 100, YRadius = 2, Color = 7; YRadius < 240;
          XRadius++, YRadius += 2 ) {
        DrawEllipse(640/2, 480/2, XRadius, YRadius, Color);
        Color = (Color + 1) & 0x0F;    /* cycle through 16 colors */
    }

    /* Wait for a key, restore text mode, and done */
    scanf("%c", &Temp);
    Regs.x.ax = 0x0003;
    int86(0x10, &Regs, &Regs);
}
```

LISTING 4

```
/* <<< This code replaces DrawEllipse in Listing 1 >>>
 *
 * Draws an ellipse of the specified X and Y axis radii and color,
 * using a fast integer-only & square-root-free approach.
 * Compiles with either Turbo C 2.0 or MSC 5.0.
 * VGA or EGA. */

/* Draws an ellipse of X axis radius A and Y axis radius B in
 * color Color centered at screen coordinate (X,Y). Radii must
 * both be nonzero */
void DrawEllipse(int X, int Y, int A, int B, int Color) {
    int WorkingX, WorkingY;
    long Threshold;
    long ASquared = (long) A * A;
    long BSquared = (long) B * B;
    long XAdjust, YAdjust;

    /* Set drawing color via set/reset */
    outpw(GC_INDEX, (0x0F << 8) | SET_RESET_ENABLE_INDEX);
                              /* enable set/reset for all planes */
    outpw(GC_INDEX, (Color << 8) | SET_RESET_INDEX);
                              /* set set/reset (drawing) color */
    outp(GC_INDEX, BIT_MASK_INDEX); /* leave the GC Index reg pointing
                                       to the Bit Mask reg */

    /* Draw the four symmetric arcs for which X advances faster (that is,
       for which X is the major axis) */
    /* Draw the initial top & bottom points */
    DrawDot(X, Y+B);
    DrawDot(X, Y-B);

    /* Draw the four arcs; set draw parameters for initial point (0,B) */
    WorkingX = 0;
    WorkingY = B;
    XAdjust = 0;
    YAdjust = ASquared * 2 * B;
    Threshold = ASquared / 4 - ASquared * B;

    for (;;) {
        /* Advance the threshold to the value for the next X point
           to be drawn */
        Threshold += XAdjust + BSquared;

        /* If the threshold has passed 0, then the Y coordinate has
           advanced more than halfway to the next pixel and it's time
           to advance the Y coordinate by 1 and set the next threshold
           accordingly */
        if ( Threshold >= 0 ) {
            YAdjust -= ASquared * 2;
            Threshold -= YAdjust;
            WorkingY--;
        }

        /* Advance the X coordinate by 1 */
        XAdjust += BSquared * 2;
        WorkingX++;

        /* Stop if X is no longer the major axis (the arc has passed the
           45-degree point) */
        if ( XAdjust >= YAdjust )
            break;

        /* Draw the 4 symmetries of the current point */
        DrawDot(X+WorkingX, Y-WorkingY);
        DrawDot(X-WorkingX, Y-WorkingY);
        DrawDot(X+WorkingX, Y+WorkingY);
        DrawDot(X-WorkingX, Y+WorkingY);
    }

    /* Draw the four symmetric arcs for which Y advances faster (that is,
       for which Y is the major axis) */
    /* Draw the initial left & right points */
    DrawDot(X+A, Y);
    DrawDot(X-A, Y);

    /* Draw the four arcs; set draw parameters for initial point (A,0) */
    WorkingX = A;
    WorkingY = 0;
    XAdjust = BSquared * 2 * A;
    YAdjust = 0;
    Threshold = BSquared / 4 - BSquared * A;

    for (;;) {
        /* Advance the threshold to the value for the next Y point
           to be drawn */
        Threshold += YAdjust + ASquared;

        /* If the threshold has passed 0, then the X coordinate has
           advanced more than halfway to the next pixel and it's time
           to advance the X coordinate by 1 and set the next threshold
           accordingly */
        if ( Threshold >= 0 ) {
            XAdjust -= BSquared * 2;
            Threshold = Threshold - XAdjust;
            WorkingX--;
        }

        /* Advance the Y coordinate by 1 */
        YAdjust += ASquared * 2;
        WorkingY++;

        /* Stop if Y is no longer the major axis (the arc has passed the
           45-degree point) */
        if ( YAdjust > XAdjust )
            break;

        /* Draw the 4 symmetries of the current point */
        DrawDot(X+WorkingX, Y-WorkingY);
        DrawDot(X-WorkingX, Y-WorkingY);
        DrawDot(X+WorkingX, Y+WorkingY);
        DrawDot(X-WorkingX, Y+WorkingY);
    }

    /* Reset the Bit Mask register to normal */
    outp(GC_INDEX + 1, 0xFF);
    /* Turn off set/reset enable */
    outpw(GC_INDEX, (0x00 << 8) | SET_RESET_ENABLE_INDEX);
}
```

All code in this issue is available through our bulletin board services or directly from *PJ* on disk. For more information see pages 8 and 88.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

COMMAND.COM, hidden treasure of undocumented DOS

Reentering the DOS shell

By Daniel E. Greenberg

While it is often claimed that MS-DOS has reached its zenith, with slow decline in popularity to follow, it is nevertheless true that some of the more fascinating features of this operating system have yet to be fully explicated. One of these is the undocumented entry to the DOS shell via INT 2Eh.

The idea behind INT 2Eh is simple. When COMMAND.COM is first loaded into memory, it sets the vector for INT 2Eh to point to code within its own resident portion. Thereafter, if we were to call INT 2Eh with DS:SI pointing to a valid command string, the root COMMAND.COM would execute the command just as if it had been entered at the DOS prompt. The vector for INT 2Eh always points to the resident portion of the root COMMAND.COM, even if we load a secondary COMMAND into memory. (If we made the secondary COMMAND "permanent"—with the little-used */p* switch—INT 2Eh would be revectored to point to the new COMMAND.COM.)

Thus, INT 2Eh does something akin to the **exec** function when we use it to load a secondary command processor. However, since INT 2Eh is always processed by the root COMMAND.COM, some interesting possibilities set it apart from the **exec** function. For example, an obstacle users often face when accessing the DOS environment is that any changes made to the environment while a secondary COMMAND.COM is active will be lost as soon as the secondary COMMAND is exited. This happens because a sec'ondary COMMAND has access only to a temporary copy of the environment, not to the permanent copy that is associated with the root COMMAND. As a result, it is impossible to make permanent changes to the environment when working out of an application's DOS shell (for example, Microsoft Word's Library Run command). If we access the environment through INT 2Eh, however, it is always the root environment—not a copy—that is altered; this holds true even with a secondary COMMAND.COM in memory.

INT 2Eh allows us to circumvent a number of other limitations that have normally been understood as inherent to DOS or COMMAND.COM. For example, a general-purpose INT 2Eh utility like the XCALL utility presented here lets versions of DOS prior to 3.3 emulate the 3.3-and-later batch **call** command, lets us created nested **for** loops, and allows the user to make use of two separate sets of environment strings loaded into memory simultaneously. It is surprising that Microsoft incorporated the foundation for these capabilities into all versions of DOS from 2.0 on but chose—for reasons not clear—to leave them undeveloped.

Unfortunately, although information has been published on the calling sequence for INT 2Eh (for example, see The Waite Group's *MS-DOS Developer's Guide*, 1989, p. 742), so far as I know nothing has been published that explains how to avoid the rather bizarre side effects that INT 2Eh is capable of creating. It is my aim here to explain how to avoid these side effects as well as how to incorporate INT 2Eh into a general-purpose utility that makes use of the root COMMAND as the reentrant resource its creators designed it to be.

SIDE EFFECTS

The easiest way to demonstrate the kinds of side effects created by this interrupt is with an example. 2Eh.ASM (Figure 1) is the source code for a simple utility that will pass any command tail to the root COMMAND.COM. The syntax for this utility is "2Eh {*command*}", where *command* is any valid DOS command or executable-file name. (Note the extreme simplicity of this approach compared with the much more thorny **exec** function. This comparative simplicity is illusory, however, as will become clear below.) To call INT 2Eh, we

Daniel E. Greenberg is a graduate student at the University of Michigan. He is interested in operating systems and the use and misuse of computer metaphors in the analysis of cognitive processes. Daniel can be reached at University Center for the Child & Family, 1007 E. Huron, Ann Arbor, MI 48104, 313-764-9466.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

point DS:SI to a command string that is terminated by a carriage return and preceded by a character count that includes the terminating carriage return. (This is the format in which DOS passes a command tail to an application at offset 80h of the program segment prefix, or PSP.)

To ensure the availability of sufficient memory to load the transient portion of COMMAND.COM into high memory, the 2Eh caller should move the stack to a safe location and release all unneeded memory before the call. Finally, because all registers except CS and IP will be trashed upon return from the interrupt, SS and SP need to be saved before the call and restored after the call.

Let's use 2Eh.COM from within a batch file (TEST.BAT) to alter the root environment:

```
echo off
2Eh set variable=one
if %variable% == one goto ok
echo ERROR, VARIABLE NOT SET
goto :end
:ok
echo VARIABLE CORRECTLY SET
:end
```

This batch file seems straightforward. 2Eh.COM uses the **SET** command to create a variable called *variable* in the root environment and sets it equal to 1. Then the environment is tested to see if *variable* has in fact been properly set. Entering **2Eh set** with no secondary COMMAND.COM active should have the same affect as just entering **SET**. Thus, if you ran test TEST.BAT without a secondary COMMAND.COM, you could expect it to issue the "Variable correctly set" message. In fact, this is not what happens. What you really see is the "Error, variable not set" message. Stranger still, if you enter **SET** after running TEST.BAT, you will see that *variable* has been correctly set.

To explain this odd behavior, we have to dig into the logic employed by COMMAND.COM to handle an INT 2Eh. COMMAND.COM processes a command passed to it through INT 2Eh just as it would if the string had been entered at the DOS prompt. When we ask COMMAND.COM to process a string passed via INT 2Eh, COMMAND.COM is in a state similar to that which exists after the termination of an application program; that is, before the command line can be processed, COMMAND.COM must attend to any open batch process or **for** process. This means that if we call INT 2Eh from an application launched by a batch file, COMMAND.COM will complete processing of the batch file before it can execute the command string pointed to by DS:SI.

This is why the error message is displayed in processing the batch file above: At the point at which the batch processor tests whether *variable* = 1, COMMAND.COM has yet to process the **SET** command passed to it via INT 2Eh; it is saving that command until after the batch file has been completely processed. Thus, *variable* will not be set to 1 until after TEST.BAT ends. (The same logic applies to INT 2Eh calls during a **for** loop—the 2Eh command is not processed until after **for** processing is complete.)

After the completion of an application or internal command, or after control is passed to it via an INT 2Eh, how does the resident portion of COMMAND.COM determine that a batch file is running? A pointer (a segment address) to a data

ILLUSTRATION BY JIM CARPENTER

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The trouble in designing a general-purpose INT 2Eh lies in what is euphemistically called "exception handling."

structure is created within the resident portion of COMMAND.COM when a batch file is opened. This data structure, which I call the "batch file data block," contains the complete path and file specification for the currently executing batch file. Other parts of the batch file data block vary by version; for example, under versions 3.0–4.01, the batch file data block also holds the command line entered at the DOS prompt to load the batch file as well as the current file-pointer location within the batch file.

The batch file data block occupies three or more paragraphs of memory, which COMMAND.COM obtains through the Allocate Memory system call (**INT 21H, AH=48**) at the time the batch file is opened (see Figure 2). When batch file execution completes, COMMAND.COM deallocates the batch file data block and resets the pointer to 0, signifying that no batch process is pending. Thus, the batch file data block pointer does double duty as a flag.

If we wish to call INT 2Eh while a batch file is open and avoid the side effect we encountered above, we must "trick" COMMAND.COM by temporarily resetting COMMAND's batch file data block pointer to 0 before issuing INT 2Eh. After the return from the interrupt, we restore the batch pointer so that COMMAND.COM will complete the batch file in the proper sequence upon our exit.

Indeed, that is the general strategy that must be followed if we are to construct a useful INT 2Eh utility. All COMMAND.COM variables that might interfere with the proper execution of the interrupt must be saved or cleared—or both—before INT 2Eh is issued and restored after the return from the interrupt. In addition to the pointer to the batch file data structure, we must save and reset a flag and a pointer to a structure that COMMAND.COM creates (in versions 3.0–4.01) in processing a **for** loop. We also have to save and later restore the flag that is used to keep track of the current state of **echo** and, in DOS 3.3 and later, the counter that keeps track of the number of times a nested **call** has been issued. Finally, since COMMAND.COM pops from the stack and saves the INT 2Eh caller's return address (returning via an intersegment jump rather than IRET) as well as the caller's PSP, and since we might use our utility to call other copies of itself (nested 2Eh calls), it is important that each INT 2Eh caller also save and later restore COMMAND.COM's copy of the previous 2Eh caller's return address and PSP.

Saving and restoring all these variables allows INT 2Eh to coexist with batch files, **for** loops, secondary COMMAND.COMs, and the **call** command and avoids the many side effects that INT 2Eh is capable of creating.

UNDOCUMENTED SHELL VARIABLES

Of course, all this begs the question of how we are to locate these COMMAND.COM variables, none of which has been documented by Microsoft. Fortunately, this is a problem that yields quite nicely to "empirical investigation"—that is, a few late-night hours with DEBUG. The structure of the data area in the various versions of COMMAND.COM is delineated in the data structures in the XCALL source code. In every version of DOS from 2.01 on, the first four words of this data structure are identical (see the "command2x_data STRUC"). Beyond that, minor differences separate the various versions.

Note that I have determined the offsets for the relevant COMMAND.COM variables only in "generic" MS-DOS and PC-DOS versions of COMMAND.COM; I have not examined the data area in copies of COMMAND.COM that accompany "enhanced" versions of DOS. (These often have in-between or odd version numbers, such as 3.31 or 3.4). I am not certain whether the structure of the relevant data areas varies significantly in these enhanced versions. Obviously, it is important to test carefully before using a utility such as XCALL with these enhanced versions. Additionally, note that in writing XCALL, I have assumed that it will run only on systems where the versions of DOS and COMMAND.COM match. If you run XCALL with an incompatible combination of DOS and COMMAND.COM, system crashes are sure to occur.

The XCALL utility relies upon the following features to locate the start of COMMAND's data area: First, INT 2Eh will always point to COMMAND.COM except in the unlikely event that some other process has revectored the interrupt. To determine if the vector to 2Eh actually points to COMMAND.COM, XCALL relies upon a crude but effective method of verifying the location of the COMMAND.COM segment: Offset 16h within a program's PSP contains the segment address of the process that "owns" that program, that is, the "parent" process. (This is an "undocumented" feature documented in Microsoft's *Programmer's Sourcebook* and elsewhere.) Since COMMAND.COM owns itself, the segment address at offset 16h should be identical to the segment address returned for the INT 2Eh vector. If these values do not match, some process other than COMMAND.COM has revectored INT 2Eh, and we cannot rely upon the vector for 2Eh to help us locate the root COMMAND. While there are a number of other methods that would reliably locate the root COMMAND.COM, XCALL simply exits with an error message if this method does not succeed in locating COMMAND.COM.

Having located the segment of the resident portion of COMMAND.COM, XCALL then proceeds to determine the offset of COMMAND's data area. The method relies upon the following: In every version of "generic" DOS from 2.1 on, the first few lines of code for the INT 2Eh handler are identical (see Figure 3). The first task the INT 2Eh handler performs is to pop and save the INT 2Eh caller's return address. The offset within the resident COMMAND.COM segment where the 2Eh caller's IP is stored can be considered, for our purposes, as the beginning of COMMAND.COM's internal data area. Since all versions of generic COMMAND.COM contain the same

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

code for the INT 2Eh handler, the machine code address for the beginning of COMMAND's data area will lie at the same offset location in the INT 2Eh handler code in all versions of the resident portion of COMMAND. We then use the offset portion of the vector for INT 2Eh to find the address of the first word of COMMAND'S data area. All elements in the COMMAND data structures delineated in the XCALL source code are then calculated as offsets from this address.

EXCEPTION HANDLING

Accounting for a number of undocumented variables whose offsets differ with each version of DOS is somewhat tedious, but the real trouble in designing a general-purpose INT 2Eh utility lies in what is euphemistically called "exception handling." The problem arises in the following situation: Under versions 3.1–4.01, a batch file that is launched through INT 2Eh is interrupted by a Ctrl-Break. Normally, when a batch file launched by the root COMMAND is interrupted, the default INT 23h handler queries the user with the "Terminate batch job?" prompt. Before issuing this prompt, however, the Ctrl-Break handler first checks COMMAND.COM's INT 2Eh flag. If the flag is set, indicating COMMAND.COM has been reentered, the user is not queried about continuing the batch file; instead, control returns to the INT 2Eh caller without the release of the batch file data block.

The implications of this are potentially disastrous. If the application that issued INT 2Eh regained control from the Ctrl-Break handler and simply terminated at that point, the root COMMAND.COM would regain control and, finding the batch block pointer still set, resume batch file execution from the point at which the user had attempted to stop it. What this means is that a batch file launched through INT 2Eh cannot be stopped by a Ctrl-Break! If you don't see the problem with that, consider a batch file containing the following:

```
pause If you do not wish to erase everything enter ctrl-break now!
del *.*
```

The INT 23h handler deals with Ctrl-Breaks issued from an INT 2Eh launched **for** loop in the same way: The interrupted **for** block is not released; instead, control returns immediately to the INT 2Eh caller. This is also similar to the approach taken by the default INT 24h handler in response to a critical error arising during an INT 2Eh–launched batch file or **for** loop. While the user will be queried with the "Abort, Retry, Ignore?" prompt, if the user chooses to abort, the batch file or **for** loop is not terminated; instead, control returns immediately to the INT 2Eh caller.

To avoid this problem, an INT 2Eh caller must test COMMAND's batch file and **for** pointers after the return from the interrupt. Since the pointers should have been cleared by the caller before the INT 2Eh, and since they are always cleared by COMMAND.COM after the completion of a batch file or **for** loop, if they are not clear after the return from INT 2Eh it can only mean that batch file or **for** execution was interrupted by a Ctrl-Break or a hard error. If this is the case, it is the responsibility of the INT 2Eh caller to deal with the batch file and **for** data blocks before termination. The simplest

FIGURE 1

```
code        segment para public 'code'
            assume cs:code,ds:code
            org 100h
begin:      jmp start

;---------data---------
stk_seg     dw    ?
stk_ptr     dw    ?

;---------code---------
start:
mov sp,offset end_here  ;move stack

push cs                 ;reduce memory
pop es
mov ah,04ah
mov bx,(offset endhere-(offset begin-100h)+15)/16
int 21h

mov si,80h              ;point ds:si to command tail
mov stk_seg,ss          ;save stack
mov stk_ptr,sp

int 2Eh                 ;reenter COMMAND.COM

cli                     ;restore stack
mov ss,cs:stk_seg
mov sp,cs:stk_ptr
sti

mov ah,4c               ;terminate
int 21h

stk_buffer  db    64 (?)
end_here    db  label

code ends
      end begin
      END
```

2EH.ASM—Simple program using INT 2Eh

FIGURE 2

```
12CF:0000 00 07 00 00 00 25 00 FF  .....%..
12CF:0008 FF FF FF FF FF FF FF FF  ........
12CF:0010 FF FF FF FF FF FF FF FF  ........
12CF:0018 FF 45 3A 5C 54 45 53 54  .E:\TEST
12CF:0020 2E 42 41 54 00 74 65 73  .BAT.tes
12CF:0028 74 00 00 00 00 00 00 00  t.......
```

Hex dump of batch file data block under DOS 3.2

FIGURE 3

```
8F06960A    CS:             ;Pop and save INT 2EH caller's
2E          POP     [0A96]  ; cs and ip.
8F06980A    CS:
58          POP     [0A98]
0E          POP     AX      ;Discard flags from the stack.

07          PUSH    CS      ;Copy command string pointed to by
                            ;  caller's DS:SI
BF8000      POP     ES      ;  to COMMAND'S PSP offset 80h.
B94000      MOV     DI,0080
F3          MOV     CX,0040
A5          REPZ
B451        MOVSW

CD21        MOV     AH,51   ;Get and save caller's PSP.
2E          INT     21
891E9A0A    CS:
B450        MOV     [0A9A],BX

8CCB        MOV     AH,50   ;Make current PSP = COMMAND's segment.
CD21        MOV     BX,CS
2E          INT     21
C7060108100 CS:

2E          MOV     WORD PTR [0B01],0081  ;Set flag to indicate 2Eh
                            ;in progress and to point to command string.
```

First few lines of code in the resident COMMAND.COM INT 2Eh handler. While the addresses vary, these line of code are otherwise identical in all versions of COMMAND.COM from 2.1 to 4.01. The code immediately following these lines is the entry point for COMMAND.COM's default termination handler.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

approach is to just terminate the interrupted batch or **for** process, and this is the approach I've taken here. More sophisticated approaches would query the user and then selectively terminate or continue current or calling batch and **for** processes.

How does an INT 2Eh caller terminate a batch file or **for** loop? Under DOS versions prior to 3.3, this a simple matter: The INT 2Eh caller resets its PSP to that of the segment of the resident portion of COMMAND.COM with the Set PSP system call (**INT 21H, AH=50h**) and then issues the Release Memory call (**INT 21H, AH=49h**) with BX equal to the segment address of the batch file data block or the **for** data block. After that, the PSP is restored and COMMAND.COM's batch block pointer or **for** pointer is reset to 0.

Under DOS versions 3.3 and later, things become more difficult: If we use INT 2Eh to launch a batch file that uses the "call" command to launch another batch file that is interrupted by a Ctrl-Break, then it is necessary to terminate not only the interrupted batch file but the calling batch file as well. Under versions 3.3+, each batch file data block holds the address of the previous calling batch file, if any, as well as the address of any active **for** loop (see Figure 4). The steps necessary to release and reset these blocks are delineated in the **release_batch_block3_3** procedure in the XCALL source code.

A final twist on the problem of exception handling arises because of differences between versions 2.x–3.0 and later versions. Under version 2.x–3.0, a Ctrl-Break entered during a batch file launched through INT 2Eh will result in the "Terminate batch job?" prompt. If the user chooses to terminate, the Ctrl-Break handler releases the batch file data block before returning control to the INT 2Eh handler. However, if the user chooses to continue batch file execution and the INT 2Eh flag is set, the Ctrl-Break handler ignores the user's request and does not continue processing the batch file; instead, control returns immediately to the INT 2Eh caller. This means that if, after the return from INT 2Eh, the batch file data block pointer is set and the version is 2.x–3.0, then the user must choose "no" in response to the "Terminate batch job?" prompt. I've designed XCALL to disregard the user's choice in this situation and simply terminate the batch file after informing the user that the batch file cannot be restarted.

USING XCALL

The syntax for using XCALL is

```
XCALL [command]
```

where *command* is any valid internal or external DOS command or executable file name. The command entered is passed to the root COMMAND.COM, where it is executed as if it had been entered at the DOS prompt with no secondary copy of COMMAND.COM active.

GRAPHICS!

Add lightning fast graphics to your programs quickly and easily through the popular PCX file format. Why reinvent the wheel? Make your programs immediately compatible with hundreds of packages from Aldus PageMaker to ZSoft's PC Paintbrush with these linkable graphic libraries.

"An exceptional product" - Programmer's Journal, Aug

NEW! Version 3.5 of the **PCX Programmer's Toolkit** gives you over 60 powerful functions to manipulate bitmapped graphics. Use Virtual screens, Super VGA modes, LIM 4.0 support, a 300 page manual, 9 utilities including screen capture and display, and the fastest routines on the market. **$195**

Need Special Effects, but caught in a GRASP?

Why create a demo when you can create the real thing? Don't be trapped in a slideshow editor or demo program when you can use **PCX Effects** for the PCX Toolkit and your favorite programming language. A Music Language and spectacular effects for exploding your graphics! **$99**

Blazing Graphics Text

With **PCX Text** you can display text with graphics as fast as it always should have been. Display characters, strings, fixed and proportional text, background transparency, and more. Includes a font editor, 85 fonts, and text utilities for blazing graphics bitmapped text. **$149**

All packages support 12 compilers for C, Pascal, Basic, Fortran, Assembly, and Clipper. All modes of the Hercules, CGA, EGA, VGA, and Super VGA adapters are supported, up through 800x600x256 (22 modes in all). Assembly Language source code is optionally available. Trademarks are property of their respective holders.

No Royalties! 30-day Money Back Guarantee.
VISA/MC/AMEX/COD/PO accepted.

G·E·N·U·S MICROPROGRAMMING
11315 Meadow Lake • Houston, Texas 77077 • (713) 870-0737

1-800-227-0918

Number **261** on the Reader Service Card

Improved Atom Manager for Microsoft Windows™

THE ATOMIZER™

The **ATOMIZER** is an atom management library that extends capability of the Windows atom management facility. The extended features, which can be enabled and disabled at runtime, provide:

- An arbitrary number of atoms (up to 16384)
- Strings of arbitrary length (up to 65535)
- Use of empty string (" ")
- Explicit access to the atom reference count
- Separate atom assignment of strings differing only in case

The **ATOMIZER** library is strictly compatible with Windows global and local atom managers and can be retrofitted to existing applications without changing the application source code. **ATOMIZER** minimizes usage of global and focal memory with sophisticated storage management algorithms.

The basic package contains the **ATOMIZER** dynamic link library and user-level documentation, with instructions and examples. The examples are supplied with their source files and auxiliary files needed to build example applications. In addition, the source package contains the library source code and an extensive program logic manual embedded via commentary in the source code.

Basic Package	3 1/2 or 5 1/4 diskette	**$39.50**
Source Package	3 1/2 or 5 1/4 diskette	**$79.50**

Please specify package and diskette size when ordering. Shipping and handling included; COD orders add $5.00. Send check or money order to:

Software Projects Group
3451 Phoebe Court
West Covina, CA 91792
(818) 913-7170

Trademark: ATOMIZER, Software Projects Group

Number **356** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

For example, to access or alter the root environment with a secondary COMMAND active, or from an application's DOS shell, enter

```
xcall set variable=value
```

After that, any command or application launched through the root COMMAND.COM will have access to a copy of the new environment, while the copy of the environment associated with the secondary COMMAND.COM will remain unaltered.

Since XCALL allows us to continue to access the root environment despite the presence of a secondary COMMAND.COM, we gain the ability, in effect, to take advantage of two independent sets of environment strings. So, for example, if we load a secondary COMMAND.COM and set the DOS path with

```
path = a;b;c;
```

we can then use XCALL to set a root path with

```
xcall path = x;y;z;
```

We can choose between two paths here. If we wish to launch an application along the "a;b;c" path, we just enter the program name from the command line. This is processed by the secondary COMMAND.COM via its copy of the environment. If we want to launch an application along the "x;y;z" path, we use XCALL to pass the program name to the root COMMAND.COM and its environment.

Traditionally, the only way to execute a series of nested batch files in versions of DOS prior to 3.3 has been through the command/c{*batchfile*} sequence. The problem with this technique is that changes made to the environment by a nested batch file cannot later be queried by the calling batch file, since the environment associated with each batch file is discarded when the corresponding COMMAND.COM exits. If we launch all our nested batch files with XCALL however,

FIGURE 4

```
1267:0000  00 01 00 00 00 00 00 0D  ........
1267:0008  00 00 00 2C 00 FF FF FF  ...,....
1267:0010  FF FF FF FF FF FF FF FF  ........
1267:0018  FF FF FF FF FF FF FF 45  .......E
1267:0020  3A 5C 46 49 52 53 54 2E  :\FIRST.
1267:0028  42 41 54 00 46 49 52 53  BAT.FIRS
1267:0030  54 0D 00 00 18 00 67 12  T.....g.
1267:0038  FF FF FF FF 00 00 00 00  ........

1272:0000  00 01 67 12 00 00 00 09  ..g.....
1272:0008  00 00 00 2D 00 FF FF FF  ...-....
1272:0010  FF FF FF FF FF FF FF FF  ........
1272:0018  FF FF FF FF FF FF FF 45  .......E
1272:0020  3A 5C 53 45 43 4F 4E 44  :\SECOND
1272:0028  2E 42 41 54 00 73 65 63  .BAT.sec
1272:0030  6F 6E 64 0D 00 00 6C 12  ond...l.
1272:0038  FF FF FF FF 00 00 00 00  ........
```

Batch blocks for FIRST.BAT and SECOND.BAT. The second byte holds the previous status of the echo flag (01 = echo on). The word at offset 2 holds the address of the batch block of the calling batch file, if any. The word at offset 4 would hold the address of a **for** processing block if the batch file had been called from within a **for** loop. Since SECOND.BAT was called by FIRST.BAT, the word at offset 2 in SECOND.BAT's block points to FIRST.BAT's block (1267h).

LISTING 1

```
;Excerpts from XCALL.ASM
;(C) COPYRIGHT 1990 DANIEL GREENBERG
;
;General-purpose interrupt 2Eh utility.
;
;Syntax:
;       xcall {command}
;command is passed to the root COMMAND.COM where it is processed
;regardless of the presence of a secondary COMMAND.COM.
;

code    segment para public 'code'
        assume cs:code,ds:code
        org 100h
begin: jmp start

;EQUATES
version_two_one         equ  020ah
version_two_one_one     equ  020bh
version_three_zero      equ  0300h
version_three_one       equ  030ah
version_three_two       equ  0314h
version_three_three     equ  031eh
version_four_zero       equ  0400h

owner_id                equ  16h        ;Offset in PSP

;Following data structures delineate important COMMAND.COM variables,
;by version:
command2x_data STRUC
TwoE_caller_ip          dw  ?
TwoE_caller_cs          dw  ?
TwoE_caller_psp         dw  ?
command_owner_id        dw  ?           ;If this COMMAND not root this
command2x_data ENDS                     ; variable holds PSP of calling
                                        ; COMMAND. Else holds root
                                        ; COMMAND's own PSP.

command4_0_data STRUC
com_twox_data_structure db  8 dup (?)   ;1st 8 bytes same structure as 2x
exit_routine4_0         dd  ?           ;EXIT transfers control to this
                                        ;  address
unkwn4_0                db  5 dup (?)
batch_block_pointer4_0  dw  ?           ;Seg address of batch file data
                                        ;  block; 0000h = no batch
                                        ;  file in progress
comspec4_0              db  64 dup (?)  ;Path to disk copy of COMMAND.COM
comspec_string_end4_0   dw  ?           ;Offset of end of comspec string
transient_command4_0    dd  ?           ;Transient COMMAND entry address
more_unkwn4_0           db  8 dup (?)
Exec_flag4_0            db  ?           ;01 = application launched
still_more_unkwn4_0     db  5 dup (?)
echo_flag4_0            db  ?           ;0 = echo off; 01 = echo on
twoE_flag_etc           dw  ?           ;0=no 2Eh in progress; 81h=COMMAND
                                        ;  reentered, 2Eh string at offset
                                        ;  81h; 0ffffh=2Eh command
                                        ;  processed,return to caller
                                        ;  pending.
even_more_unkwn4_0      db  9 dup (?)
for_flag4_0             db  ?           ;01 = for loop in progress
for_block_pointer4_0    dw  ?           ;Seg address of "for" process
                                        ;  block
batch_counter4_0        dw  ?           ;Incremented on each nested batch
                                        ;  "call"
command4_0_data ENDS

;This structure delineates first 6 bytes of
;batch file data block (ver 3.3+):
batch_block STRUC
unknown                 db  ?
prev_echo_flag          db  ?           ;0 = echo off; 01 = echo on
previous_batch_blk      dw  ?           ;Calling batch file block, if any
previous_for_blk        dw  ?           ;Calling for loop data block,
batch_block ENDS                        ;if any

for_flag_address        dw  ?
batch_ptr_address       dw  ?

;We save COMMAND's info on previous 2Eh caller here. Later restore
;this to COMMAND.COM so on our exit COMMAND will return control
;to previous caller:
previous_2e_psp         dw  ?
previous_2e_ip          dw  ?
previous_2e_cs          dw  ?
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
previous_2e_flag        dw  ?
two_flagoffset          dw  ?

;We save COMMAND's flags and pointers here prior to 2Eh call
;  and restore this info to COMMAND after call.
;  This allows COMMAND.COM to continue normal
;  execution after our exit:
old_batch_block         dw  ?
old_batch_fileptr       dd  ?           ;Used in DOS 2.x only
old_batch_counter       dw  ?
old_echo_flag           db  ?
old_for_flag            db  ?
old_for_block           dw  ?           ;Not used in DOS 2.x

stk_ptr                 dw  ?
version                 dw  ?

first                   db  0           ;0=no previous active int 2Eh
error                   db  0           ;0=ok, 1=error during xcalled
                                        ;  batch file or for loop

batch_terminate_mes     db  0dh,0ah,"Cannot restore XCALL'ed batch file "
                        db  "after Ctrl-break or critical error.",0dh,0ah
                        db  "Batch file will be terminated.",0dh,0ah,"$"


;Note on register usage: First subroutine sets es:di equal
;  to segment:offset
;  of data area for root COMMAND.COM.  After that, es:di are
;  preserved across subroutines.

reduce_memory:
        int 21h

        push di                         ;di=offset of COMMAND's data area.
        mov stk_ptr,sp                  ;Save_stack

        int 2eh                         ;Reenter root COMMAND.COM.

        mov ax,cs                       ;Restore registers & stack.
        cli
        mov ss,ax
        mov sp,cs:stk_ptr
        sti
        mov ds,ax
        pop di
        pop es                          ;es:di=seg:offset of
                                        ;  COMMAND's data area
        call check_error                ;Hard error or ctrl-c
                                        ;  during batch file?
        cmp error,01                    ;If so, need to flush
                                        ;  disc buffers.
        jnz x1
        mov ah,0dh
        int 21h                         ;Disk reset.

x1:     call restore_batch_state        ;Restore previous batch file,
                                        ;  if any.
        call reset_for                  ;Restore for loop, if necessary
        cmp first,01
        jz x2

        call restore_previous_2e        ;Restore COMMAND's info on
        jmp x3                          ;  previous XCALL if any. (This
                                        ;  gurantees that COMMAND.COM
                                        ;  will return
                                        ;  control to previous XCALL
                                        ;  after our exit.)

x2:     call reset_echo_flag            ;Restore echo state

x3:     mov ah,4ch                      ;Return to COMMAND.COM.
        int 21h


;Restore_batch_state restores COMMAND.COM's batch block pointer. Call with
;ES = COMMAND'S seg & DI = offset of COMMAND's data area.

restore_batch_state proc near

        mov bx,batch_ptr_address        ;Reset COMMAND's batch ptr to
what it
        mov ax,old_batch_block          ; was before INT 2Eh
        mov word ptr es:[bx],ax
```

continued

this problem is eliminated, since each batch file has access to the same (root) environment. Thus, XCALL allows us to make use of the following kind of execution sequence:

```
one.bat:
xcall two.bat

        two.bat:
        xcall three.bat
                etc.
```

With each iteration, a new batch file data block is allocated, while the contents of the preceding one continue to be preserved in memory. With such a technique, we can nest batch file calls to almost unlimited depth.

XCALL also allows nesting to be carried out in **for** loops. The following, for example, will execute properly:

```
xcall for %a in (*.asm) do
xcall for %b in (*.exe) do
xcall del *.obj
```

(Nested **for** loops will not work with versions of DOS prior to 3.0, however, because in these versions the **for** data block is not allocated dynamically via the Allocate Memory DOS call but, rather, occupies a fixed buffer in the COMMAND.COM segment. If we try to use INT 2Eh to nest **for** under version 2.x, each iteration of **for** simply overwrites the previous and the nesting is destroyed. A simple enhancement that would solve this problem would be to have XCALL preserve the entire contents of the COMMAND 2.x **for** buffer in its own segment before calling INT 2Eh.)

XCALL is designed so that if you wish an XCALLed command to be redirected or piped, you enclose the entire command tail in quotation marks. This ensures that DOS will not process the redirection command before it is passed to XCALL. XCALL then strips the first and last quotation marks before passing the string to the root COMMAND.COM. For example,

```
xcall "set >environ.doc"
```

would redirect the output from SET to environ.doc. On the other hand,

```
xcall set >environ.doc
```

would redirect the output from XCALL to environ.doc.

CONCLUSION

INT 2Eh is no EXEC function substitute. The complexities of handling a variety of undocumented variables makes it a poor choice when an application needs to provide the user with general access to the DOS command processor. On the other hand, since it adds reentrant capabilities to COMMAND.COM, INT 2Eh provides programmers with a powerful resource with which to accomplish tasks previously thought impossible under DOS. For this reason, as well as from what we learn about COMMAND.COM in the course of exploring it, INT 2Eh is one of the more fascinating of the undocumented DOS functions.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        cmp version,version_three_zero
        jae rbs_3x

        mov ax,word ptr old_batch_fileptr  ;If version 2.x we also have to
        mov word ptr es:[bx+2],ax          ;  restore the file pointer.
        mov ax,word ptr old_batch_fileptr+2
        mov word ptr es:[bx+4],ax
        ret

rbs_3x:
        cmp version,version_three_three
        jb rbs1
                                    ;If 3.3+ we also have to restore the
        mov ax,old_batch_counter    ;  batch counter.
        mov word ptr es:[di].batch_counter3_3,ax

rbs1: ret

restore_batch_state endp

;Reset_for resets the for flag and (in versions 3.0+) the pointer to the for
;processing block.  Call with ES = COMMAND'S segment.
reset_for proc near
        mov al,byte ptr old_for_flag
        mov bx,for_flag_address
        mov byte ptr es:[bx],al
        cmp version,version_three_zero  ;No for block in ver. 2.x
        jge rf1
        ret

rf1:    mov ax,old_for_block
        mov word ptr es:[bx+1],ax
        ret
reset_for endp

;Restore_previous_2e restores COMMAND'S info on previous 2Eh caller.  This
;allows COMMAND.COM to return control to previous caller on our exit. Call
;with ES:DI = seg:offset of COMMAND'S data area.
restore_previous_2e proc near
        mov ax,previous_2e_psp          ;Reset COMMAND 2Eh info
        mov es:[di].twoe_caller_psp,ax  ;  on caller's psp
        mov ax,previous_2e_ip
        mov es:[di].twoe_caller_ip,ax   ;Reset COMMAND 2Eh info
        mov ax,previous_2e_cs           ;  on caller's ip
        mov es:[di].twoe_caller_cs,ax   ;Reset COMMAND 2Eh info
                                        ;  on caller's cs
        mov bx,twoE_flagoffset          ;Restore COMMAND 2Eh flag.
        mov ax,previous_2e_flag
        mov word ptr es:[bx],ax
        ret
restore_previous_2e endp

;Reset_echo_flag restores COMMAND's echo flag (2.1 - 3.2). to what it was
;prior to INT 2Eh.  Call with ES:DI = seg:offset of COMMAND'S data area.
reset_echo_flag proc near
        cmp version,version_three_three
        jge ref2
        mov al,byte ptr old_echo_flag
        cmp version,version_three_two
        jge ref_3_2
        cmp version,version_three_zero
        jge ref_3_0

;version 2 - restore echo to what it was prior to xcall:
        mov byte ptr es:[di].echo_flag2,al
        ret

ref_3_0:                                    ;(For 3.0 & 3.1)
        mov byte ptr es:[di].echo_flag3_0,al
        ret

ref_3_2:
        mov byte ptr es:[di].echo_flag3_2,al

ref2:
        ret
reset_echo_flag endp

even
stack_space             db  64 dup  (?)
endhere                 label byte

;ERROR ROUTINES
error1mes               db "Requires DOS 2.1 or later.",0dh,0ah,"$"
error2mes               db "Error: Interrupt 2Eh vector has been "
                        db "altered.",0dh,0ah,"$"

start label near

        mov ah,30h                  ;Get version.
        int 21h
        xchg ah,al
        cmp ax,version_two_one      ;Exit if not 2.1 or later.
        jb error1

        mov version,ax

        call check_vector           ;Get 2Eh vector - points to COMMAND?
        jc error2                   ;If not, exit.

        mov di,es:[bx+3]            ;INT 2eh vector offset + 3 = address
                                    ; of COMMAND's data area.
                                    ;es:di now = COMMAND'S data area.

        call check_command_string   ;Check for redirection.
        call save_previous_2eh      ;Check if another 2Eh in progress
        call save_batch_state       ;Check if batch file in progress
        call save_echo_state        ;Save contents of echo flag
        call save_for_state         ;Check if for loop in progress

        mov si,80h                  ;Point SI to command string.

        mov sp,offset endhere       ;Move stack to safe area so we can
                                    ; reduce memory allocation.

        push es                     ;Save COMMAND's segment
        mov ax,cs                   ;Set up to reduce memory allocation
        mov es,ax
        mov ah,04ah
        mov bx,(offset endhere-(offset begin-100h)+15)/16
        jmp reduce_memory

;Check_vector determines if INT 2Eh vector points to COMMAND.COM. If so, it
;returns ES:BX pointing to segment:offset of 2Eh handler, otherwise it sets
;the carry flag.
check_vector proc near
        xor ax,ax                   ;Get interrupt 2Eh vector.
        mov es,ax
        mov bx,2eh * 4
        les bx,es:[bx]              ;es:bx = segment:offset.

        mov ax,es                   ;Does 2Eh vector point
                                    ; to COMMAND.COM?
        cmp word ptr es:[owner_id],ax   ;  (COMMAND.COM owns itself.)
        jz cv1
        stc                         ;If not, set carry.
        ret
cv1:    clc
        ret
check_vector endp

;This routine determines if an INT 2Eh is already in progress and, if so,
;  saves the offset and contents of COMMAND.COM's 2Eh flag and address
;  information on previous 2Eh caller.  If an INT 2Eh is not in progress a
;  flag ("first") is set to indicate this. Call with ES=COMMAND.COM segment
;  and DI = offset of COMMAND's data area.
save_previous_2eh proc near
        mov bx,es:[bx+27h]          ;Offset for INT 2Eh vector + 27h =
        cmp word ptr es:[bx],0000h  ; address of COMMAND's 2Eh flag.
        jz sp1                      ;Is it set?
        mov ax,es:[bx]

        mov twoe_flagoffset,bx      ;Save address of 2Eh flag.
        mov previous_2e_flag,ax     ;Save contents of 2Eh flag.
        mov ax,es:[di].twoe_caller_psp  ;Save previous 2Eh caller's psp.
        mov previous_2e_psp,ax
        mov ax,es:[di].twoe_caller_ip   ;Save previous 2Eh caller's ip.
        mov previous_2e_ip,ax
        mov ax,es:[di].twoe_caller_cs   ;Save previous 2Eh caller's cs.
        mov previous_2e_cs,ax
        ret

sp1:    mov first,01                ;No, we are first 2Eh caller.
        ret
save_previous_2eh endp

;Save_batch_state determines if a batch file is in progress and, if so,
;saves the offset and contents of COMMAND's batch file data block pointer.
;The pointer is then set to zero. Call with ES = COMMAND.COM segment &
;DI = offset of COMMAND's data area.
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;Save_batch_state determines if a batch file is in progress and, if so,
;saves the offset and contents of COMMAND's batch file data block pointer.
;The pointer is then set to zero. Call with ES = COMMAND.COM segment &
;DI = offset of COMMAND's data area.
save_batch_state proc near
        cmp version,version_three_zero
        jae sbs1

;For 2.10 & 2.11:
        mov bx,offset batch_block_pointer2  ;Save address of batch pointer
        add bx,di
        mov batch_ptr_address,bx
                                        ;Is a batch process pending?
        cmp word ptr es:[bx],0000h
        jz sbs_end
        mov ax,word ptr es:[bx]
        mov old_batch_block,ax          ;Save current batch pointer.
        mov word ptr es:[bx],0000h      ;Zero block pointer.
        mov ax,word ptr es:[di].batch_fileptr   ;Save batch fileptr.
        mov word ptr old_batch_fileptr,ax
        mov ax,word ptr es:[di].batch_fileptr+2
        mov word ptr old_batch_fileptr+2,ax
        mov word ptr es:[di].batch_fileptr,0000h  ;Zero batch fileptr.
        mov word ptr es:[di].batch_fileptr+2,0000h
        ret

;for version 3 and later:
sbs1:
        mov bx,offset batch_block_pointer3  ;Save offset of batch pointer
        add bx,di
        mov batch_ptr_address,bx
        cmp word ptr es:[bx],0000h      ;Batch process pending?
        jz sbs_end
        mov ax,es:[bx]
        mov old_batch_block,ax          ;Save current batch pointer.
        mov word ptr es:[bx],0000h      ;Reset COMMAND's batch pointer.

        cmp version,version_three_three ;If 3.3+ save "call" counter.
        jb sbs_end

        mov ax,word ptr es:[di].batch_counter3_3
        mov old_batch_counter,ax
                                        ;Zero batch call counter.
        mov word ptr es:[di].batch_counter3_3,0000h

sbs_end:
        ret
save_batch_state endp

;Save_echo_state saves the echo flag for versions 2.1 - 3.2.
;Call with ES = COMMAND.COM segment & DI = offset of COMMAND's data area.
save_echo_state proc near
        cmp version,version_three_three
        jge ses4
        cmp version,version_three_two
        jge ses2
        cmp version,version_three_zero
        jge ses1

;for version 2.x:
        mov cl,byte ptr es:[di].echo_flag2
        jmp ses3

ses1:                                   ;for 3.0 & 3.1
        mov cl,byte ptr es:[di].echo_flag3_0
        jmp ses3

ses2:                                   ;for 3.2
        mov cl,byte ptr es:[di].echo_flag3_2

ses3:
        mov old_echo_flag,cl

ses4:
        ret
save_echo_state endp

;Save_for_state saves the offset address of COMMAND's "for" block, checks
;if a "for" loop is being executed and, if so, saves and then clears the
;for flag and the pointer to the for process block. Call with
;ES=COMMAND.COM segment and DI=offset of COMMAND's data area.
save_for_state proc near
        cmp version,version_four_zero
```

continued

Tired of the limitations of MS-DOS?
Thinking about switching to another OS?
Well DON'T, now there is:

PRO-CLI v4.1

CLI is a tiny, intelligent front & back end processor that enhances the MS-DOS system kernel. CLI provides many features of advanced OS's, like VMS & UNIX, to MS-DOS without the tremendous costs and additional training.

CLI-Types: ALIASs, LOGICALs, SYMBOLs, LEXICAL-FUNCTIONs and STREAMs are available to the command line, batch programs, & all existing application programs. CLI is 100% MS-DOS compatible, and NO changes are required in any software to reference any CLI-Types.

Some of the features offered by PRO-CLI include:

- batch programs with numeric & string processing
- facilities for source code and project management
- transparent task-to-task communications
- file & device management from the command line
- command line editing, history, and recall
- flexible system installations & re-configurations
- user configurable PRO-CLI resources

For the first time, designers, developers, and managers can coordinate software development & production efforts without the major restrictions of MS-DOS.

Call or Write for Additional Information!

Trademarks: VMS of D.E.C. MS-DOS of Microsoft UNIX of AT&T

Alphax Systems, Inc.
1005 Sussex Blvd.
Broomall, PA 19008
(215) 543-2658

Price: $ 149
S/H: $ 6

Major Credit Cards Accepted

Number 345 on the Reader Service Card

Multitasking

Do the multitasking systems you are considering using seem a bit like overkill or need an awful lot of memory? Maybe all you really need is a system which can run more than one DOS program concurrently. If this sounds like your situation, consider MultiDos Plus. Flexible preemptive scheduling of multiple tasks with full access to MS-DOS or PC-DOS. LIM 4.0 support. For more information please call for a free brochure.

Nanosoft
800-678-2141
508-651-0091
FAX 508-655-8860
13 Westfield Road
Natick MA 01760

Number 36 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PERISCOPE® IV

THE Real REAL-TIME SOURCE-LEVEL DEBUGGER

With Periscope Model IV, you can debug your software while it runs at full speed, something *no* software-only debugger can do.

You can also examine the execution history of a hardware interrupt in Periscope IV's real-time trace buffer, something *else* no software-only debugger can do.

When The Going Gets Tough... Periscope IV helps you debug most any program, but is specially designed to debug programs that software-only debuggers can't. Model IV users use it to debug TSRs, ISRs, device-drivers, DOS, BIOS, communications software, real-time data acquisition programs, multitasking software, network software, keyboard drivers, disk caching software, systems software, spread sheets, EMM products, and so forth.

"Periscope is my #1 favorite program... I would rather change my editor than my debugger," writes Phil Mayes, who used Periscope IV to track down some very elusive bugs causing intermittent corruption.

Periscope IV provides source-level support for popular PC compilers and linkers, such as those produced by Microsoft, Borland, and others, and it runs on most any 80286 or 80386 at speeds up to 25MHz.

The new Remote Debugging feature lets you use Periscope IV to debug programs running on IBM PS/2s and compatibles. The optional Plus board keeps Periscope from using any of the lower 640K, so Periscope can't be overwritten and doesn't use the memory your program needs.

You can try Periscope IV for ten days at no charge before you buy it. Call our toll-free number for details.

FREE 10-day Evaluation

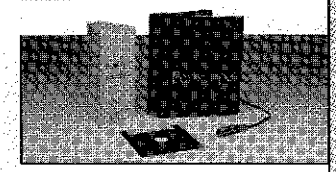
Periscope IV prices range from $2195 for a 16 MHz 80286 system to $2995 for a 25 MHz 80386 system. The optional 512K Periscope PLUS board is $400.

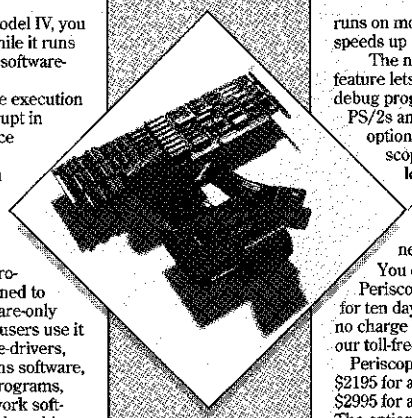
Given the value of your time, can you afford *not* to try it?

Call Toll-Free Today for More Information or to Order Your Periscope: 800-722-7006

In the UK call Roundhill Computer Systems at 0672 84 535. In Germany call H+B EDV at 07542 6353 or ComFood at 02534 7093. In Sweden call LinSoft at 013 124780.

The Periscope manual, software, breakout switch and quick-reference card are included with Model IV.





"The Model IV hardware really makes Periscope shine, especially when you've got timing-related problems. I can now track down changing pointers and altered buffers on my 386. I've been using it to debug Crosstalk® Mk. 4 and there's just no better way to do it."

JEFF GARBERS
Director of Software Development
Crosstalk Communications

The Periscope Company, Inc.
1197 Peachtree St., Plaza Level
Atlanta, GA 30361
404/875-8080 • FAX 404/872-1973

Number **6** on the Reader Service Card

DOS & OS/2
Free Demo Diskette Available

Q. How many programmers does it take to maintain a MAKE dependency file?

A. NONE! *If you use* VersiMAKE™

VersiMAKE™ is a full-featured MAKE utility that includes:

- **Dependency Generation**
  Derives your system's dependencies, through analysis of its C and MASM source files. Say goodbye to manual maintenance of MAKE dependency files!
- **Wild Card File Name Matching**
  Analyzes an entire collection of source files with a single statement.
- **Nested Include Files**
  Handles standard C and MASM "include" conventions, and the INCLUDE environment variable.
- **Powerful Macro Facilities**
  Built-in macros, user-defined macros, and environment variables.
- **Analytical Reports**
  Shows the entire Include file hierarchy for each source file analyzed, and all of the parent source files for each Include file.

Q. How many programmers does it take to trace a symbol thru your system?

A. ONE! *If you use* VersiCREF™

VersiCREF™ is a unique utility that creates a Master Cross-Reference of your entire system.

- **Multi-Lingual**
  Handles C, assembler, or both.
- **Flexible**
  File names with line numbers, or file names alone. Global and local symbols, or globals alone.
- **Powerful**
  Easily handles systems containing 100 source files or more.

VersiMAKE™	$125
VersiCREF™	$75
Both	$150

(Outside U.S., add $5 for shipping and handling)

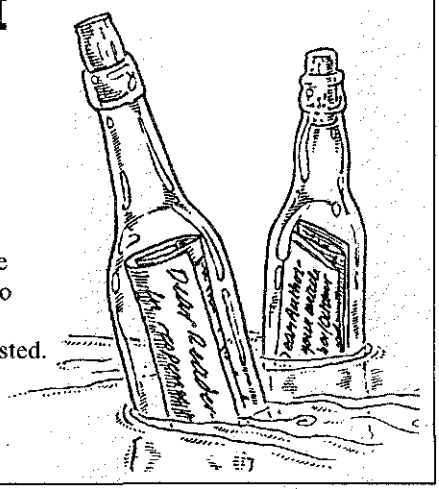
800-344-4096
(In NJ, 609-779-0784)
MC/VISA/AMEX accepted

SUMMIT INFORMATION SYSTEMS, INC.
496 N. Kings HWY Suite 130
Cherry Hill, NJ 08034

Number **158** on the Reader Service Card

Free Product Information

Use our Reader Service Card (found between pages 88 & 89) to get valuable information on the products and services that appear in ***Programmer's Journal.*** Just circle the number on the card that corresponds to the number at the bottom of each advertisement in which you are interested. And please remember to mention ***Programmer's Journal*** when you contact our advertisers.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        cmp version,version_three_zero
        jge sf_3_0
        jmp sf_2_1

sf_4_0:                                 ;4.0 & 4.01
        mov ax,offset for_flag4_0       ;Get offset of for flag.
        add ax,di
        mov for_flag_address,ax

        mov cl,es:[di].for_flag4_0      ;Get contents of for flag.
        cmp cl,00                       ;If 0 then no for loop in progress.
        jz sf2
        mov ax,es:[di].for_block_pointer4_0     ;Get for data block pointer.

        mov byte ptr es:[di].for_flag4_0,00h    ;Clear for flag &
                                        ;  block pointer.
        mov word ptr es:[di].for_block_pointer4_0,0000h

sf1:
        mov old_for_flag,cl             ;Save for flag.
        mov old_for_block,ax            ;Save for block.
sf2:    ret

sf_3_3:                                 ;3.3
        mov ax,offset for_flag3_3       ;Get offset of for flag.
        add ax,di
        mov for_flag_address,ax

        mov cl,es:[di].for_flag3_3      ;Get contents of for flag.
        cmp cl,00                       ;If 0 then no for loop in progress.
        jz sf2
        mov ax,es:[di].for_block_pointer3_3     ;Get for data block pointer.

        mov byte ptr es:[di].for_flag3_3,00h    ;Clear for flag &
                                        ;  block pointer
        mov word ptr es:[di].for_block_pointer3_3,0000h
        jmp sf1

sf_3_2:                                 ;3.2
        mov ax,offset for_flag3_2       ;Get offset of for flag.
        add ax,di
        mov for_flag_address,ax

        mov cl,es:[di].for_flag3_2      ;Get contents of for flag.
        cmp cl,00h                      ;If 0 then no for loop in progress.
        jz sf2
        mov ax,es:[di].for_block_pointer3_2     ;Get for data block pointer.
        mov byte ptr es:[di].for_flag3_2,00h    ;Clear for flag &
                                        ;  block pointer.
        mov word ptr es:[di].for_block_pointer3_2,0000h
        jmp sf1

sf_3_0:                                 ;3.0 & 3.1
        mov ax,offset for_flag3_0       ;Get offset of for flag.
        add ax,di
        mov for_flag_address,ax
        mov cl,es:[di].for_flag3_0      ;Get contents of for flag
        cmp cl,00h                      ;If 0 then no for loop in progress
        jz sf2
        mov ax,es:[di].for_block_pointer3_0     ;Get for data block pointer.
        mov byte ptr es:[di].for_flag3_0,00h    ;Clear for flag &
                                        ;  block pointer.
        mov word ptr es:[di].for_block_pointer3_0,0000h
        jmp sf1

sf_2_1:                                 ;2.1 & 2.11
        mov ax,offset for_flag2         ;Get offset of for flag.
        add ax,di
        mov for_flag_address,ax
        mov cl,es:[di].for_flag2        ;Get contents of for flag.
        cmp cl,00h                      ;If 0 then no for loop in progress.
        jz sf2

        mov byte ptr es:[di].for_flag2,00h      ;Clear the for flag
        xor ax,ax                       ;No for data blk in ver 2.x.
        jmp sf1

save_for_state endp

code ends
        end begin
        END
```


The Heap Expander™ version 2.05

Now your programs can have virtually unlimited heap space using expanded memory, extended memory, disk space, or any combination of the three. And it's all transparent. The Heap Expander's initialization code checks the system's resources and uses whatever is available.

only $79.95*

- Uses LIM-standard expanded memory if present.
- Uses AT-style extended memory if present.
- Swaps data to disk as needed.

Libraries and Source Code for:

- Turbo C
- Microsoft C 4.0 and 5.0

Requires an IBM PC, XT, AT, or close compatible with MS-DOS or PC-DOS version 2.0 or above

MC/VISA/COD call
1-800-248-1045 x 100 (US)
1-800-952-5560 x 100 (Idaho)

*CA residents add 6 1/2% sales tax. Foreign customers, please add $4.00 for shipping and handling.

The Tool Makers
P. O. Box 2151
Santa Cruz, CA 95063
(408) 458-0690

Number 166 on the Reader Service Card

GUIDO

C Library for Graphical User Interface

GUIDO is a powerful set of C user interface functions for use in a graphical environment. GUIDO's high level objects make it easy to build intuitive front ends for your software.

- Pull-down, pop-up, vertical and horizontal menus.
- Movable, overlapping, windows, data entry, text and list boxes.
- Radio buttons, percent bars, prompt boxes and more.
- Turbo C, Microsoft C and Quick C.
- EGA and VGA support.
- Price $249 with source $499

Tel. - 201-762-6965

SOUTH MOUNTAIN SOFTWARE™
1-800-451-6174

Number 10 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C CODE FOR THE PC

source code, of course

	MS-DOS File Compatibility Package (read & write MS-DOS file systems on non-MS-DOS computers)	$750
	CSource Application Program Generator by MBAC (includes all source code; generator & libraries)	$500
	dB2c (dBase-to-C translator; includes db_Files for C and db_Tools for C)	$325
	CQL Query System (SQL retrievals on B-trees plus windows)	$325
	GraphiC 5.0 (high-resolution, DISSPLA-style scientific plots in color & hardcopy)	$325
	Drasch Clisp with Crules (Lisp library and programming environment with rule processing capability; natural language example)	$300
	PC Curses (Aspen, Software, System V compatible, extensive documentation)	$290
	C-Data Manager (object-oriented data management, persistent objects from runtime definitions, network and entity models)	$250
	MEWEL (extensible window and event library by Magma Software; message-passing & object-oriented; SAA-compatible; dialog editor)	$250
	TurboTEX (Release 2.0; HP, PS, dot drivers; CM fonts; LaTEX; MetaFont)	$250
	PC PostScript (complete PostScript interpreter (ROM Version 47.0A), 80286/386 only, many device drivers, optimized graphics, fast)	$250
	db_File & db_Retrieve by Raima (B-tree and network database with SQL query and report writer; multi-user $475)	$245
	Greenleaf Communications Library (interrupt mode, modem control, XON-XOFF; specify compiler)	$225
	CDirect (multi-user hashed file manager; variable length fields, binary or ASCII data, alternate keys)	$210
NEW!	SilverComm (complete asynchronous communications library)	$210
	Installation Toolkit 3.0 (interactive script development program; many special cases covered)	$195
	QuickGeometry Library (large collection of mathematics, graphics, display & DXF subroutines for CAD/CAM/CAE/CNC)	$170
	CBTree (B+tree ISAM driver, multiple variable-length keys)	$165
	TurboGeometry (library of routines for computational geometry, Version 3.0)	$160
	AT BIOS Kit (roll your own BIOS with this complete set of basic input/output functions for ATs)	$160
	WKS Library Version 2.01 (C program interface to Lotus 1-2-3, dBase, Supercalc 4, Quatro, & Clipper)	$155
	OS/88 (industrial-strength U**x-like operating system, many tools, cross-development from MS-DOS)	$150
	Cephes Mathematical Library (over 100 high-quality, double-precision scientific functions)	$150
	ME Version 2.1 (programmer's editor with C-like macro language by Magma Software; Version 1.31 still $75)	$140
	Vmem/C (virtual memory manager; least-recently used pager; dynamic expansion of swap file)	$140
	Turbo G Graphics Library (all popular adapters, hidden line removal)	$135
NEW!	Rogue Wave Vector & Matrix Classes (inc. C++ overloadings for standard operators, matrix inversion & FFT; Zortech or GNU C++)	$125
	Power Search by Blaise Computing (regular-expression compiler; generates machine code on the fly)	$120
	Install 2.3 (automatic installation program; user-selected partial installation; CRC checking)	$120
NEW!	B-Strings (dynamic string handling; cut, copy, paste, search, user input, etc.; non-fragmenting memory management)	$105
	TE Editor Developer's Kit (full screen editor, undo command, multiple windows)	$105
	Minix Operating System (Version 1.3; U**x-like operating system, includes manual)	$105
Updated!	PC/IP (CMU/MIT TCP/IP for PCs; Ethernet, Appletalk & NETBIOS drivers, RVD, gateways)	$100
	B-Tree Library & ISAM Driver (file system utilities by Softfocus)	$100
	The Profiler (program execution profile tool)	$100
	QC88 C compiler (ASM output, small model, no longs, floats or bit fields, 80+ function library)	$90
Cheaper!	Booter Toolkit (floppy disk bootstrap routines, DOS file system, light-weight multitasking, windows, fast memory management)	$85
	Otter 1.0 (beautiful theorem-prover by Bill McCune; includes manual & two books by Wos; complete starter kit)	$80
	JATE Async Terminal Emulator (includes file transfer and menu subsystem)	$80
	MultiDOS Plus (DOS-based multitasking, intertask messaging, semaphores)	$80
NEW!	HY-PHEN-EX (a hyphenator for American English with over 4,800 rules)	$75
	Make (macros, all languages, built-in rules)	$75
	eval() (C function to evaluate ASCII infix expression string; 17 built-in functions)	$75
	XT BIOS Kit (roll your own BIOS with this complete set of basic input/output functions for XTs)	$75
	Professional C Windows (lean & mean window and keyboard handler)	$70
	Heap Expander (virtual memory manager using expanded memory, extended memory, and disk space)	$65
	Quincy (interactive C interpreter)	$60
	Symtab/Ptree (general-purpose symbol table/parse tree construction and management package; specify Symtab or Ptree)	$60
	Coder's Prolog (Version 3.0; inference engine for use with C programs)	$60
	Async-Termio (Unix V compatible serial interface for MS-DOS; stty, ioctl, SIGINT, etc.)	$55
	Backup & Restore Utility by Blake McBride (multiple volumes, file compression & encryption)	$50
NEW!	Floppy TAR (TAR backup and restore on MS-DOS devices; direct access to non-standard devices)	$50
	SuperGrep (exceptionally fast, revolutionary text searching algorithm; also searches sub-directories)	$50
	REGX Plus (search and replace string manipulation routines based on regular expressions)	$50
	OBJASM (convert .obj files to .asm files; output is MASM compatible)	$50
	Multi-User BBS (chat, mail, menus, sysop displays; does not include Hayes modem driver)	$50
	LaplaceB (LaPlace polynomials, real and complex)	$50
	CLIPS (rule-based expert system generator, Version 4.3; advanced manuals available)	$50
	Kier DateLib (all kinds of date manipulation; translation, validation, formatting, & arithmetic)	$45
	Fortran-to-C Translator by Polyglot (Fortran-IV-like Fortran to ugly C; plan to adapt to your own flavor of Fortran)	$40
	DES Encryption & Decryption (2500 bits/second on 4.77 MHz PC for on-the-fly encryption at 2400 baud)	$40
	FlexList (doubly-linked lists of arbitrary data with multiple access methods)	$40
	Virtual Memory Manager by Blake McBride (LRU pager, dynamic swap file, image save/restore)	$40
	Heap I/O (treat all or part of a disk file as heap storage)	$40
Updated!	Bison & BYACC (YACC workalike parser generators; documentation; no restrictions on use of BYACC output)	$35
	PC-XINU (Comer's XINU operating system for PC)	$35
	RXC & EGREP (Regular Expression Compiler and Pattern Matching; RXC makes finite state machine from regular expression)	$35
	CCALC (handy extended-precision calculator; real and complex models; many built-in functions)	$30
	GNU Awk & Diff for PC (both programs in one package)	$30
Updated!	6-Pack of Editors (baker's half-dozen public domain editors for use, study & hacking; includes microEmacs 3.10 & Stevie, a vi clone)	$30
	Crunch Pack (14 file compression & expansion programs)	$30
	Pascal P-Code Compiler & Interpreter or Pascal-to-C Translator (Wirth standard Pascal)	$25
	FLEX (fast lexical analyzer generator; new, improved LEX; official BSD Version 2.1 with docs)	$25
	List-Pac (C functions for lists, stacks, and queues)	$25
NEW!	Using C++ Library (the code from the book by Bruce Eckel and then some; Zortech 2.0 compatible)	$25
	A68 (68000 cross-assembler)	$20
	XLT Macro Processor (general purpose text translator)	$20
	Data	
	Moby Pronunciator (150,000 words & phrases encoded with full IPA pronunciation & emphasis points; 900 distinguished by part-of-speech)	$160
	Moby Part-of-Speech (200,000 words and phrases described by prioritized part(s)-of-speech)	$120
	Moby Hyphenator (150,000 words fully hyphenated/syllabified)	$105
	Moby Words (500,000 words & phrases, 9,000 stars, 15,000 names)	$65
	Smithsonian Astronomical Observatory Subset (right ascension, declination, & magnitude of 258,997 stars)	$60
	U. S. Cities (names & longitude/latitude of 32,000 U.S. cities and 6,000 state boundary points)	$35
	The World Digitized (100,000 longitude/latitude of world country boundaries)	$30
	KST Fonts (13,200 characters in 139 mixed fonts: specify TEX or bitmap format)	$30
Updated!	Interactive Computer Ephemeris (high-precision moon, sun, planet & star positions; USNO (no source) & Downey 4.8 (C source))	$30
	U. S. Map (15,701 points of state boundaries)	$15

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3409 USA

info@acw.com

Voice: (512) 258-0785
BBS: (512) 258-8831
FAX: (512) 258-1342

Free surface shipping for cash in advance **For delivery in Texas add 7%** **MasterCard/VISA**

Number 30 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Cellular automatons for graphics programming

Order from chaos

By William S. Hall

Cellular automatons provide an interesting vehicle to demonstrate the basic techniques of graphics programming in Windows. In this article, I show how the use of colored rectangles and a rule for changing states can be used to generate attractive spiral patterns from chaotic beginnings. Along the way, I illustrate basic techniques for using the Windows graphics device interface (GDI) to set the mapping mode, create pen and brush objects, draw simple figures, and read the color table of an output device.

Of course, the best way to run the accompanying Wincel program is on a machine having a color display. Ideally, your Windows setup can show 16 colors. If you own a VGA or EGA, you are nodding your head yes, right? Wrong. The standard Windows EGA/VGA display drivers support only 8. However, 8 colors still provide some nice results. I have even made provisions for those of you with monochrome screens. By dithering, the 8 basic colors can be emulated well enough to make this program interesting.

Note that, lacking access to the hardware, I have not tried the program on 256-color displays. Since each color represents a state, 256 of them may be simply too much for the number of points in the automaton's universe (fixed at 50). By limiting the number of states to 12 or 14, the resulting colors will probably be rather dark and not well differentiated. If the results are not satisfactory, perhaps you can use the source code provided to create a suitable version.

If your display has more than 256 colors, you still won't see more than that number: To save space, the states are stored in an array of bytes; so, the maximum state value is 255.

Wincel has been marked to operate with future versions of Windows. Stay tuned for announcements.

This kind of program will execute best on an 80386-based machine with fast video. Since the entire screen is redrawn on each display update, the program tends to hog the processor for large amounts of time. However, I have made some provision for yielding to Windows for you to be able to use the mouse or keyboard to change parameters or kill the program.

All of the material here was inspired by A. K. Dewdney's "Computer Recreations" column from the August 1989 issue of *Scientific American* (see references). That article contains a number of nice color photographs.

William S. Hall is a senior engineer at Novell, in San Jose, California. He specializes in Windows software.

BACKGROUND

A *cellular automaton* is an infinite grid of cells in which each cell is in one of a number of possible states. A cell changes its state in response to a clock tick according to a given set of rules. The particular creation used here, called "Cyclic Space," was discovered by David Griffeath, of the University of Wisconsin. It is quite simple: If a cell is in state k at a certain time, it must consume any adjacent cells that are in state k - 1 before the next time tick. An adjacent state is consumed by changing its state from k - 1 to k. Thus, the rule is similar to a food chain where a fish eats a smaller fish while that fish is eating yet a smaller one.

Based on this rule, a space of random distributions, called "debris," can be turned into a stable distribution of angular spirals known as "demons." As the metamorphosis progresses, droplets of the same state appear in the debris. Then, crystalline spirals begin to appear during the defect stage, feeding as they evolve and competing for space with other spirals. Finally, the demon state is reached. Each of the intermediate states of debris, droplets, and defects are metastable: They last for many cycles before giving way to the next phase.

Computer implementation of cyclic space was discovered by using the Cellular Automaton Machine (CAM) developed by Norman Margolus and Tommaso Toffoli of MIT (see references). Apparently, they have developed a board that can fit into a PC. Using special hardware, CAM can compute a 256-by-256 cellular array at sixty times a second.

Cyclic space can be simulated (more slowly, of course) in

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

software in the following way. An *n*-state machine has states 0 to *n* - 1. The state below 0 is *n* - 1 itself; that is, if a cell is in state 0, it consumes an adjacent *n* - 1 state. The states themselves are stored in a rectangular array. The cells adjacent to a given cell are those to the left, right, above, and below. Left and right and top and bottom edges of the array are respectively identified so that the space is effectively a torus.

Let **Neighbor** be a procedure that successively computes the indices of the four adjacent points of a toroidal array. An algorithm in C describing the process of consuming and displaying states is as follows:

```
BYTE new[MAXPOINTS][MAXPOINTS], old[MAXPOINTS][MAXPOINTS];
register int i, j;
int k, l, n;
do {                    // consume neighboring points
    for (i = 0; i < MAXPOINTS; i++)
        for (j = 0; j < MAXPOINTS; j++)
            for (n = 0; n < 4; n++) {
                Neighbor(n, i, j, &k, &l);
                if (old[k][l] == ((old[i][j] + 1) % numstates)) {
                    new[i][j] = old[k][l];
                    break;
                }
            }
    for (i = 0; i < MAXPOINTS; i++)  // display the results
        for (j = 0; j < MAXPOINTS; j++) {
            display(new[i][j]);
            old[i][j] = new[i][j];
        }
} while (TRUE);
```

A brief study of the above shows what is happening:

Assume at the start that the new and old arrays contain the same random assortment of states. The old array is scanned. If an adjacent point can eat the given point, then its state is incremented by one (modulo the number of states) and placed into the new array. When the scan is complete, the new array is displayed and then copied back into the old one.

THE WINCEL PROGRAM

Wincel (which is contained on this issue's code disk) implements in Windows the cellular automaton described above. After running the program, pull down the Control menu and select New to generate a random array of colored rectangles on the screen—and then watch the action. At any time, you can choose New to start another run.

You can cause the action to pause and resume by clicking the left mouse button. If the window is minimized, the automaton continues to compute, but only the icon is drawn.

Colors are used to give a visual representation of the states. Using the Parameters menu, you can adjust the number of states to a value up to the maximum number of colors provided by your display (or 8 if the screen is monochrome). Also, you can set the number of iterations between displays to any value between 1 and 100. If you want to watch points of adjacent states actually consuming each other, you must set the value to 1. The default value is 5.

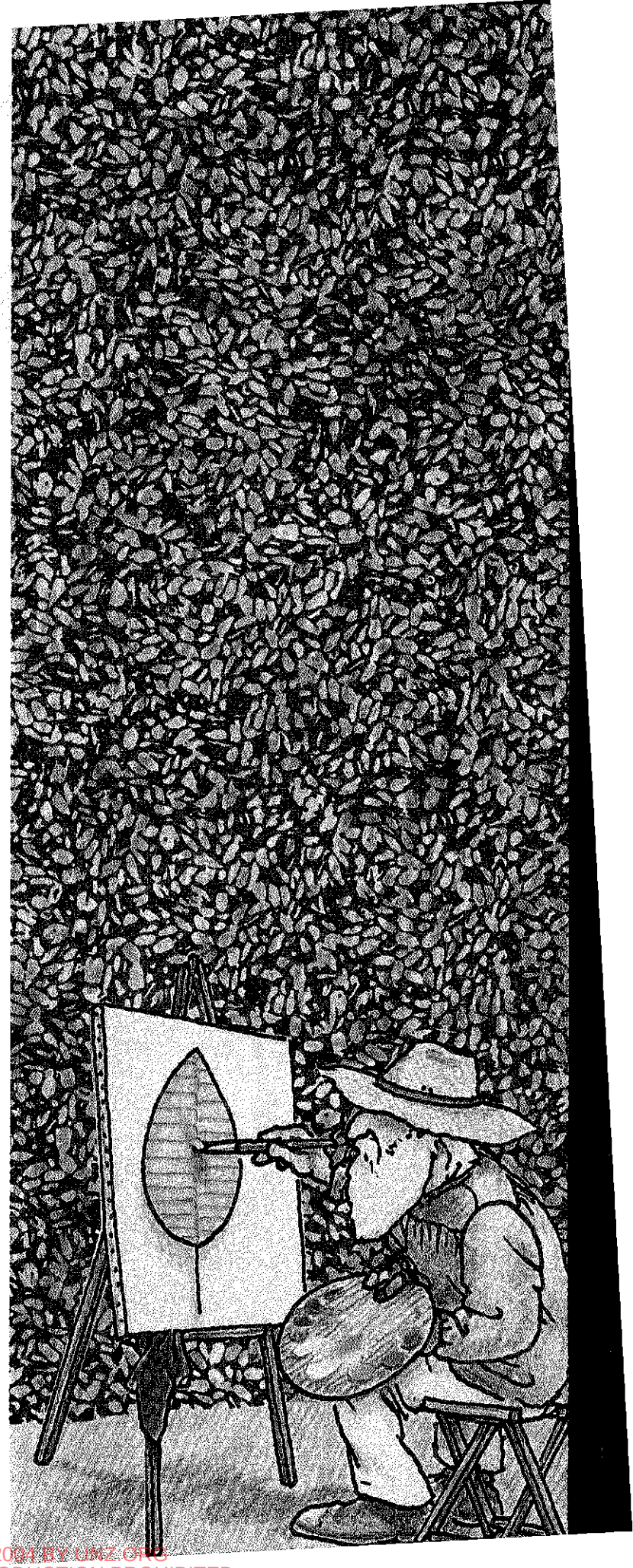


ILLUSTRATION BY JIM CARPENTER

PRODUCED 2004 BY UMZ OHG
ELECTRONIC REPRODUCTION PROHIBITED

The number of states defaults to the number of colors available if the former is 8 or less. If you have a 16- or 256-color display, the default value is 14. I chose this value because I found that with a larger number of states, the equilibrium state was mostly one color. There is no doubt a relationship between the number of states and the number of points that determines the final appearance. In Wincel, 50 points are used. You can, of course, recompile the program to use a larger number of points, but you may have to extend the memory model to accommodate very large arrays.

The Parameters dialog box also provides a list in RGB format of the device's color table. I apologize for not showing the colors themselves. However, writing this sort of custom control requires quite a bit of effort. I shall try to do so for you in a later issue. Until that time, let me explain how to read the color table.

A color in Windows is described by a double-word quantity in which the low-order byte describes the intensity of red, the next byte describes green, and the one following that describes blue. The high-order byte is arbitrary but is usually set to 0. Thus, the possible number of logical colors is 2^{24}. However, most output devices can realize only a subset of the maximum number. Some typical values for 8- and 16-color drivers are given in Table 1. This chart will help you to interpret which colors are adjacent when you look at the color-table list box.

TABLE 1

Typical eight-color table (VGA/EGA)

Color	Value in Hex
Black	000000
Bright Red	0000FF
Bright Green	00FF00
Bright Yellow	00FFFF
Bright Blue	FF0000
Bright Magenta	FF00FF
Bright Cyan	FFFF00
White	FFFFFF

Typical sixteen-color table

Color	Value in Hex
Black	000000
Red	000080
Green	008000
Yellow (Brown)	008080
Blue	800000
Magenta	800080
Cyan	808000
Gray	808080
Light Gray	C0C0C0
Bright Red	0000FF
Bright Green	00FF00
Bright Yellow	00FFFF
Bright Blue	FF0000
Bright Magenta	FF00FF
Bright Cyan	FFFF00
White	FFFFFF

Although the number of colors on a device may be small, Windows attempts to extend the number of colors by dithering whenever a filled figure, such as a rectangle, is drawn. A figure is filled by means of an 8-bit-by-8-bit object called a *brush*. By altering the bits in the brush according to certain rules, colors not in the device's color table can be visually approximated. In particular, on a monochrome device, there are patterns corresponding to the bright colors given in Table 1. This is why Wincel will give reasonable results even on a monochrome screen.

IMPLEMENTATION OF WINCEL

Two problems immediately present themselves in getting the cellular automaton to run in Windows. The first is to find a suitable representation of the states to provide an attractive display using the full client area on the screen. The second is to recognize that considerable computation time will be required for the display and to make appropriate provision for this and other applications to read Windows messages.

The easiest and most reliable way to yield the processor periodically is to use a standard **PeekMessage** loop, as described in one of my articles in an earlier issue of *PJ* (see references). Basically, if there are no messages requiring attention, the first part of the algorithm above, consuming neighboring states, is executed. Then a WM_USER message is posted back to the program. On the next call through the message loop, the WM_USER message will be retrieved. In response to the message, the second part of the algorithm is carried out. Listing 1 shows the corresponding code.

Even so, once the code to paint the window begins, several seconds elapse before it is completed. So, a **Yield** call (which correctly does nothing) has been added after each row is drawn. By these means, it is possible to switch to other applications, but they will run quite slowly. It does not matter—you will be too busy watching the pretty pictures to do anything else.

THE DRAWING CODE

Each state in the cellular automaton can be interpreted as an index into a color table. The display algorithm uses this information to show the new array on the screen. However, it was not initially obvious how to show a 50-by-50 array of points in a meaningful way.

In a first attempt, I tried simply to light a 50-by-50 array of pixels, but on an EGA or VGA, the resulting display was simply too small to be interesting. What was needed was a way to use the entire client area of the window. The answer lay in using colored rectangles and an anisotropic mapping mode. Let's begin with the easy part: drawing the rectangles. Refer to the **ShowRect** routine in the middle of Listing 2 for the following discussion.

First, recall that before drawing can take place to an output device, a display context (DC) must be obtained. Since all drawing in Wincel takes place in response to a WM_PAINT

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

message, a DC is already available. When it is retrieved, a DC has a number of default attributes (see Table 2), which are usually changed before drawing takes place. For example, if we drew a rectangle with the default DC, then the border would be drawn with the current pen (black) and filled with the current brush (white). But we need a colored one; furthermore, we want the border to be drawn with the same color. So, the first step is to create a brush of the desired color. Then it must be selected into the display context. At that time, we retrieve the handle to the old brush, because later we must restore it before returning the DC to Windows.

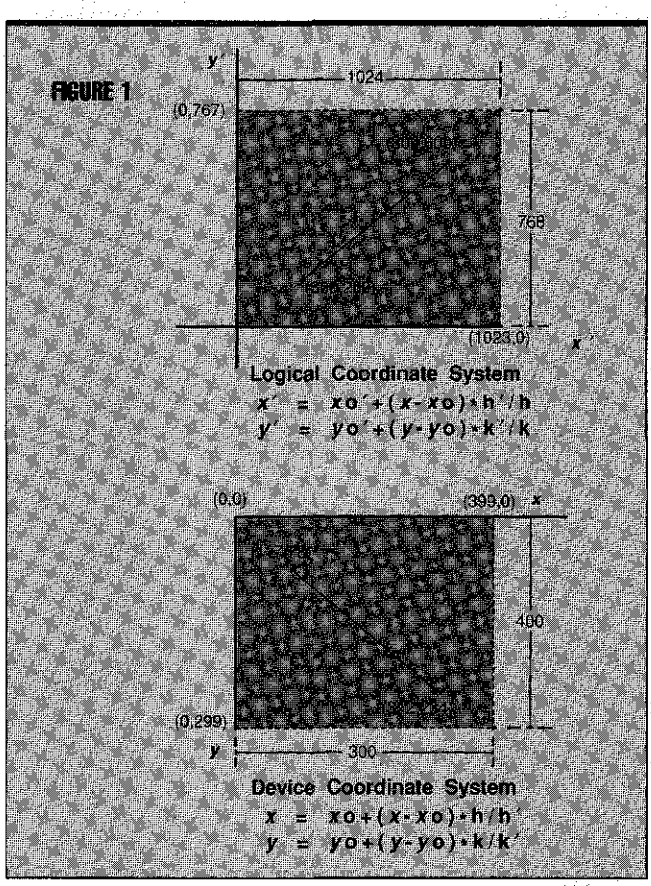
Similarly, we create a pen having the same color, select it into the DC, and retrieve the old pen. Then the rectangle is drawn where specified by its upper left and lower right coordinates, and the old pen and brush are returned to the DC. (This effectively deselects the color pen and brush from the DC.) Finally, the pen and brush objects we created are destroyed. It is important to follow the order of operations exactly as described.

Recall that the state of a given cell is an index into the device's color table. The table itself is quite easily accessed by means of the Escape function. However, if the device is monochrome (determined during initialization), an appropriate color value, based on the usual color table for an eight-color device, is constructed.

POSITIONING THE RECTANGLES

Regardless of the size of Wincel's window, we would like to have the array of colored rectangles exactly fill the client area when they are drawn. To achieve this, we take advantage of mapping modes and coordinate transformations. Let us begin by examining Window's logical and physical coordinate systems.

All drawing in Windows is done by reference to logical coordinates, which are defined by a standard, two-dimensional, Cartesian coordinate system extending from -32,767 to 32,767 in both the horizontal and vertical directions. The origin is at (0,0), and the positive x and y directions are,



LISTING 1

```
/* Peek Message loop and support functions */

    while (TRUE) {
        if (PeekMessage((LPMSG)&msg,NULL,0,0,PM_REMOVE)) {
            if (msg.message == WM_QUIT)
                break;
            TranslateMessage((LPMSG)&msg); // translate keydown/up to
                                           // chars.
            DispatchMessage((LPMSG)&msg);  // call the appropriate
                                           // window proc.
        }
        else if (go) {
            EatPoints();        // change states of points.
            PostMessage(hWndMain, WM_USER, 0, 0L); // copy and
                                                   // display states.
        }
    }


/* consume neighboring points */

static void NEAR EatPoints()
{
    register int i, j;
    int k, l, n;

    for (i = 0; i < MAXPOINTS; i++) {
        for (j = 0; j < MAXPOINTS; j++) {
            for (n = 0; n < 4; n++) {
                Neighbor(n, i, j, &k, &l);
                if (old[k][l] == ((old[i][j] + 1) % numstates)) {
                    new[i][j] = old[k][l];
                    break;
                }
            }
        }
    }
}

// for a given index i, compute the neighboring indices

static void NEAR Neighbor(int pos,register int i,register int j,
                          int *k,int *l)
{
    switch(pos) {
        case 0:
            *k = (i == 0 ? MAXPOINTS - 1 : i - 1);
            *l = j;
            break;
        case 1:
            *k = (i == MAXPOINTS - 1 ?  0 : i + 1);
            *l = j;
            break;
        case 2:
            *k = i;
            *l = (j == 0 ? MAXPOINTS - 1 : j - 1);
            break;
        case 3:
            *k = i;
            *l = (j == MAXPOINTS - 1 ? 0 : j + 1);
            break;
    }
}
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TABLE 2

Display Context Defaults (partial listing)

Background color	White
Background mode	Opaque
Brush	White
Clipping region	Entire client area is the update region clipped as appropriate.
Current position	(0,0)
Device origin	Upper left corner of client area.
Drawing mode	Copypen
Font	System font
Mapping mode	MM_TEXT
Pen	Black
Text color	Black
Viewport origin	(0,0)
Viewport extent	(1,1)
Window origin	(0,0)
Window extent	(1,1)

respectively, to the right and up. The device coordinate system, however, has its origin at the upper left corner of the display context, and positive x and y directions are, respectively, to the right and down. Each point corresponds to a pixel position on the device.

Relating the two systems are a set of transformation equations that determine where the logical points will actually appear on the device. In Figure 1, we show the essential ideas in an example. On the left, we imagine a logical screen of 1024 by 768 on which we have drawn a line extending from (300,200) to (800,600). On the right is a window whose client area is 400 pixels wide and 300 pixels deep. By using the coordinate transformations given under the figure, we can determine where the line will actually fall.

Entering into the computation are four parameters: the logical origin (xo′,yo′), the viewport origin (xo,yo), the logical extents (h′,k′), and the viewport extents (h,k). Each of these parameters can be set by an appropriate function call to Windows once a display context has been retrieved. In the example, the logical and physical origins are at (0,0), the logical extents are 1024 in the x direction and 768 in the y direction, and the viewport extends, horizontally, 400 units to the right and, vertically, 300 units down. For quite a different effect, change the viewport origin to (0,299) and the vertical extent to -300, and recalculate. Now you will find that the line runs from (117,221) to (312,65) and thus, except for the scale distortion, the line has the same general look as its logical image.

Note that the extents enter into the calculation only as ratios. They express, in fact, the amount of compression or expansion (scaling) in the coordinate directions. The mapping mode actually determines if these are independent of each

Prize catch for Windows, OS/2 and DOS developers.
INFO WORLD
1989
PRODUCT OF THE YEAR
BYTE
AWARD OF DISTINCTION
TECHNICAL EXCELLENCE
PC MAGAZINE
Bugs take note. MultiScope™ is the only debugger that is CodeView™ compatible and provides up to 13 views of programs.
The only debugger to offer run-time and post-mortem debugging. The only debugger to provide Windows, Presentation Manager and character mode user interfaces.
The only debugger to incorporate 386 virtual machine and ICE technologies from Nu-Mega, makers of Soft-ICE™ and MagicCV.™
The only debugger "that lets you focus on the problem to be solved rather than the tool used to solve it." –IEEE Software 11/89
And the only debugger to win top industry awards from PC Magazine, Info World and BYTE.
DOS–$179, DOS & Windows–$379, OS/2–$449.
For your free trial-version, please call 800-999-8846, (415) 323-4286, or FAX (415) 323-4186. For European pricing, please call ++46-13-63189.
MULTISCOPE™ DEBUGGER
The World's Best Debuggers
MultiScope, Inc., a Logitech Company
MultiScope is a trademark of MultiScope, Inc. CodeView is a trademark of Microsoft Inc. Soft-ICE and MagicCV are trademarks of Nu-Mega.
Circle Number 19 on the Reader Service Card.

PRODUCED 2006 BY JRC.ORG
ELECTRONIC REPRODUCTION PROHIBITED

other. For example, in MM_ISOTROPIC mode, a rectangle that is logically 200 units on a side will also appear as a square on the screen. In MM_ANISOTROPIC mode, which is the one we use here, extents in the horizontal and vertical directions are not coupled, and the 200-by-200 square will be mapped into a rectangle that may not have sides of equal length.

When a display context is first retrieved, both the logical and viewport origins are set at (0,0) and both extents are (1,1); hence, logical and physical units coincide. Referring to Figure 1 again, a line drawn from (0,0) to (399,299) in the logical system will fall on the device from the upper left corner to the lower right, as you would expect. It is only when mapping mode, origins, and extents are changed that you must start thinking about the two different systems when drawing, filling, or displaying text.

Now we can describe how the rectangles are positioned. It is quite simple: Suppose Wincel's automaton is described by an *n*-by-*n* array of states. To represent each state, we use a colored rectangle five units on a side; hence, we need 5 * *n* points in the *x* and *y* directions in the logical system to show the whole automaton. If now the viewport extents are chosen to be the client area of the window, then the *n* rectangles in the logical system will be visible in Wincel's window, regardless of its actual size. The code to carry out this program is show in the paint routine at the top of Listing 2.

Windows provides a powerful set of graphical primitives that allow complex figures to be drawn in a device-independent way. Of course, you must make the initial effort of learning enough about Windows to be able to use the system, and you need to develop an effective technique for yielding the processor during extensive operations. However, by applying some of the ideas presented here, you should be able to use Windows to portray other automatons as well as to study some of the more interesting geometrical items found in the area of fractal geometry, bifurcation diagrams, and strange attractors.

For additional ideas, you might examine the recent book by Roger Stevens (see references). It is a rich source of material whose realization in Windows would be an interesting task.

REFERENCES

Dewdney, A. K. "Computer Recreations: A Cellular Universe of Debris, Droplets, Defects and Demons," *Scientific American*, Vol. 261, No. 2 (1989), pp. 102–105.

Fisch, R., et al. *Cyclic Cellular Automata in Two Dimensions*, Birkhauser (in press).

Hall, William S. "A Substitute for the Missing TYPE Command in Windows," *Programmer's Journal*, Vol. 7.3 (1989), pp. 64–73.

Stevens, R. T. *Fractal Programming in C*, M&T Publishing Inc., 1989.

Toffoli, T., and N. Margolus. *Cellular Automata Machines*, MIT Press, 1988.

UNDERSTANDING
C
VIDEOTAPES

Learn the techniques used by experienced C programmers.

Get guaranteed results from this informative 3-volume, 6-hour videotape course. Loaded with examples that cover every nook and cranny of ANSI C. Handy reference manual included.

$74.95 postpaid (plus tax in Texas). Money-back guarantee. No extra charge for C.O.D. orders.

To order C.O.D., call:
800-762-5515
(Canadian orders: $79.95 in U.S. funds)

AUTOMATION EDUCATION
2309 Royce Drive
Arlington, TX 76016-1232

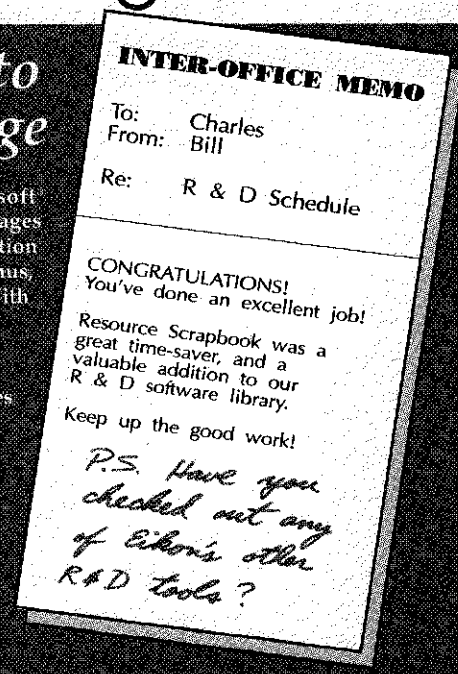
Number **207** on the Reader Service Card

Windows Programmers

We want you to get this message

Resource Scrapbook™ for Microsoft Windows™ quickly and easily manages clipboard data and various application resources – such as cursors, icons, menus, strings, dialog boxes, and more. With Resource Scrapbook, you can:

- ✓ Build entire libraries of resources and other clipboard data
- ✓ Convert selected Windows resources to OS/2™ PM™ equivalents
- ✓ Capture images of any Windows screen with ScreenCapture
- ✓ Select from over 100 ready-to-use resources in a sample file
- ✓ Examine and extract resources from your .EXE or .RES files with Resource X-Ray (included).


INTER-OFFICE MEMO
To: Charles
From: Bill
Re: R & D Schedule
CONGRATULATIONS!
You've done an excellent job!
Resource Scrapbook was a great time-saver, and a valuable addition to our R & D software library.
Keep up the good work!
P.S. Have you checked out any of Eikon's other R&D tools?

Suggested U.S. List Price: $225

Add power and versatility to your resource management!

ORDER TODAY!
Call (415) 349-4664

Eikon Systems, Inc.
989 E. Hillsdale Blvd, Suite 260
Foster City, California, USA 94404

EIKON

Number **349** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 2

```
/* update the screen */
static void NEAR MainWndPaint(HWND hWnd, HDC hDC)
{

    register int i,j;
    HCURSOR hcur, holdcur;
    RECT rect;

  /* draw the icon */
    if (IsIconic(hWnd)) {
        GetClientRect(hWnd, (LPRECT)&rect);
        Rectangle(hDC, 0,0,rect.right, rect.bottom);
        TextOut(hDC,2,rect.bottom/3,(LPSTR)szIcon,strlen(szIcon));
    }
    else if (start) {
        // put up hourglass cursor
        hcur = LoadCursor(hInst, IDC_WAIT);
        holdcur = SetCursor(hcur);

        // how big is the screen
        GetClientRect(hWnd, &rect);

        /* Set up a logical screen of MAXPOINTS x MAXPOINTS
           rectangles each of size 5 units. */
        SetMapMode(hDC, MM_ANISOTROPIC);
        SetWindowOrg(hDC, 0, 0);
        SetWindowExt(hDC, 5 * MAXPOINTS, 5 * MAXPOINTS);
        SetViewportOrg(hDC, 0, 0);
        SetViewportExt(hDC, rect.right, rect.bottom);

        // show the points
        for (i = 0; i < MAXPOINTS; i++) {
            for (j = 0; j < MAXPOINTS; j++)
                ShowRect(hDC, i, j, (int)new[i][j]);
            Yield();    // let someone else read a message
        }
        SetCursor(holdcur);     // restore the old cursor
    }
}

/* draw a rectangle on the screen using the display's color table */
static void NEAR ShowRect(HDC hDC, int i, int j, int index)
{
    HBRUSH hbr, hbrold;
    HPEN hpen, hpenold;
    register short x,y;
    DWORD color;

    // get the color for this index.
    // If monochrome, make one up
    if (monochrome)
        color = 0xff0000 * ((index & 4) >> 2) +
                0x00ff00 * ((index & 2) >> 1) +
                0x0000ff * (index & 1);
    // if color, read the color table of the device.
    else
        Escape(hDC, GETCOLORTABLE, NULL, (LPSTR)&index, (LPSTR)&color);

    // create and select a brush for the interior
    hbr = CreateSolidBrush(color);
    hbrold = SelectObject(hDC, hbr);

    // we want the same color on the boundary,
    // so create the appropriate pen.
    hpen = CreatePen(0, 0, color);
    hpenold = SelectObject(hDC, hpen);

    // draw the rectangle at the given position
    x = 5 * j;
    y = 5 * i;
    Rectangle(hDC, x, y, x + 5, y + 5);

    // select out and kill the drawing objects
    SelectObject(hDC, hbrold);
    SelectObject(hDC, hpenold);
    DeleteObject(hbr);
    DeleteObject(hpen);
}
```


A professional workstation environment for OS/2™

Hamilton C shell™

The superior alternative to the standard OS/2 command processor. Faithfully recreates and updates the entire C shell language as described in the Berkeley 4.3 *UNIX® Programmer's Manual*. Created explicitly for OS/2 using modern incremental compiler technology. Not one line ported from or created on anything but OS/2. Very high performance. Extensive support for multi-threading.

Features: Very long command lines: 64KB = 10x the UNIX limit • Fully recursive filename wildcarding: complex patterns, multiple directory-level matches • Fully nestable `if`, `foreach` and other control structures • Variables, arrays and a wide range of expression operators and built-in functions • History for recalling and editing past commands • Faster interrupts, I/O redirection and piping • Command substitution: use the output of one command as command-line arguments to another • Aliases and shell procedures • Filename completion • Arrow keys • Hashing to speed searches of the `PATH` directories • Background threads and processes, `ps` and `kill` • Numerous professional-quality utilities.

Requires OS/2 1.1 or later. All executables will run properly inside a Presentation Manager window. Not copy-protected.

$350.00. Unconditional satisfaction guarantee.

Phone 508-358-5715. MasterCard and Visa accepted.

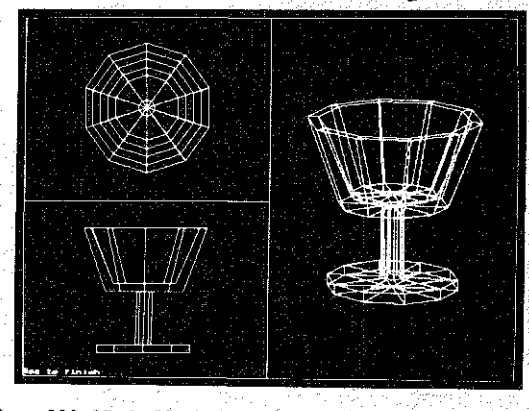
Hamilton Laboratories
13 Old Farm Road, Wayland, MA 01778–3117

Number **334** on the Reader Service Card

THE ULTIMATE CAD/CAM ENGINE

TurboGeometry Library 3.0!

Helps You build it right THE FIRST TIME!



Over 300 2D & 3D device independent geometric routines. Surfacing; Spherical & Solid Geometry; Tangents; Curves; Hidden Line; Polygons; Perspectives; 2D & 3D Clipping; Link List Library & more. In C or Pascal, w/source. 400 pg manual. $199.95, Foreign $225.00. Prices incl S&H. 30 day guarantee. VISA, MC, PO, Chk.

2116 E. Arapaho Rd.
Suite 487
Richardson, TX 75081
USA

Disk Software, Inc.

FAX or Phone
214-423-7288
Toll Free
1-800-635-7760

Number **229** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FOR IMMEDIATE PRESS RELEASE

PINNACLE PUBLISHING, INC.

Pinnacle announces the release of Graphics Server™ SDK for Windows

May 1, 1990

For more information contact:
David Johnson
Pinnacle Publishing, Inc.
PO BOX 8099
Federal Way, WA 98003
1 800 231-1293 or
206/941-2300

FEDERAL WAY, WASH.—Pinnacle Publishing, Inc., today announced the May 1, 1990 availability of Graphics Server (SDK), a Software Development Kit for Graphics Server, a charting and graphing extension for Windows™ 2.1 and above.

Graphics Server provides 12 different graph types, including 3-D Pie and Bar Charts, and can support virtually all Windows applications through DDE (Dynamic Data Exchange) and DLL (Dynamic Link Libraries). The Graphics Server SDK includes reference materials and the 'C' API to Graphics Server. The cost of the Graphics Server SDK is $495.00 and is available directly from Pinnacle Publishing. The SDK includes five copies of Graphics Server for distribution by the developer.

A Windows application may request Graphics Server to display a particular graph type. Upon receipt of the request and data to graph, Graphics Server creates a new window and displays the graph requested in the window. Window management and resizing of the graph window are managed by Graphics Server without additional overhead from the application.

"Graphics Server provides an immediate benefit to all Windows application developers by allowing the developer to concentrate on core technology development without having to worry about including graphics," said David Johnson, Vice President. "A simple interface between the application and Graphics Server is all that is necessary for the application to provide graphics to its user."

Also being released is a retail product, the Graphics Server, which will be available beginning June 1, 1990 and will include interfaces to Excel™, Superbase 4™, and Word for Windows™. Additional application interfaces are scheduled to be added throughout 1990 at nominal pricing to registered users of Graphics Server. The retail price for Graphics Server will be $149.00. A copy of Graphics Server is required for each copy of Windows. Site licenses are available.

While Graphics Server works with all Windows applications, developers may distribute a run-time version of Graphics Server that works only with their particular application for as little as $5.00 per copy in large volume purchases. Further details are available from Pinnacle.

###

Graphics Server is a trademark of Pinnacle Publishing, Inc. Other brand and product names mentioned herein are trademarks or registered trademarks of their respective holders.

Number **27** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

By Chip Little

Mouse messages

Processing mouse input under OS/2 Presentation Manager

Efficient processing of mouse input is essential to most Presentation Manager (PM) applications, but it is not adequately explained in the OS/2 documentation. Here we look at Hittest.c, a PM application that demonstrates the handling of mouse input and some of the basics of screen painting.

The program is simple: Three child windows are placed inside the main application window. Each child window receives messages from the user when the pointer is positioned over the window. Subtle differences in the message processing affect the way in which these messages alter screen appearance.

As the application demonstrates, it is possible for a window selectively to ignore or redirect mouse messages. This concept is called "hit-testing."

PUSHING THE BUTTONS

When you run the application, the initial screen looks like the one in Figure 1. When you place the mouse pointer in the white area and press button 1, the background becomes pink, as shown in Figure 2. If you position the pointer over the black window and press button 1, nothing happens. However, if you move the pointer to the center window and press button 1, the screen returns to its original appearance. Finally, when you position the pointer over the third child window and click button 1, the stripes in the child window change from vertical to horizontal, as in Figure 3.

Chip Little is an independent consultant who has worked with OS/2 since the initial development effort. He is now working with NCR on OS/2 2.0 development.

A TYPICAL PM APPLICATION

An examination of the source code reveals how three sibling windows can handle mouse input in different ways. The main procedure is typical of all PM applications. Before an application can process input, it must obtain an anchor handle and a message queue. This is done using calls to **WinInitialize** and **WinCreateMsgQueue**. The **WinCreateHeap** call, which is not required for all applications, is used to obtain a heap handle. This heap handle will be used for dynamically allocating memory later in the program.

The **WinRegisterClass** call is used to register the window procedure that is to be called for all windows of a defined class. A window procedure is the routine that processes messages to a window. In this case, the window procedure **ClientWindowProc** is used to process messages to all windows of class "Hittest." These windows have the CS_SIZEREDRAW bit set in the class-style parameter; this forces the window to be repainted when it is resized.

WinCreateStdWindow is used to create the frame window. A frame window is actually a collection of several windows. The frame flags are used to enable various child windows that are associated with the frame window. Associated with this frame window are a title bar, a system menu, wide borders, and minimize and maximize windows. The FCF_SHELLPOSITION flag tells the shell to determine the initial screen location of the window.

FCF_TASKLIST is used to enter the program title in the task manager's task list. The call itself creates a client window of type "Hittest." The client window is the "big white area" in the center, where all the drawing activity takes place. WS_VISIBLE creates a visible window (default is invisible). WS_CLIPCHILDREN excludes child windows from being updated when the window is redrawn. (We will see why later.) "HwndClient" is the handle of the client window.

The **while** loop that follows is the heart of a PM application. **WinGetMsg** waits for messages to appear on the specified queue. All window messages have associated message IDs, typically WM_*. All messages with IDs other than WM_QUIT are dispatched to the appropriate window procedure via **WinDispatchMsg**. A WM_QUIT message causes **WinGetMsg** to return false, which forces an exit from the **while** loop. The three **WinDestroy*** functions destroy the heap, frame window, and message queue. **WinTerminate** releases the anchor-block handle.

ClientWindowProc, a typical window procedure, takes four arguments:

- *hwnd* is the handle of the window receiving the message.
- MSG is the message ID (the WM_* mentioned earlier).
- *mp1* and *mp2* are message-specific parameters.

GETTING THE MESSAGE

Window procedures are typically nothing more than large switch statements. The various cases process messages of interest to the window. Other messages are passed along to the default window procedure, **WinDefWinProc**. Since window procedures are called by OS/2, the entry points must be exported as in hittest.def.

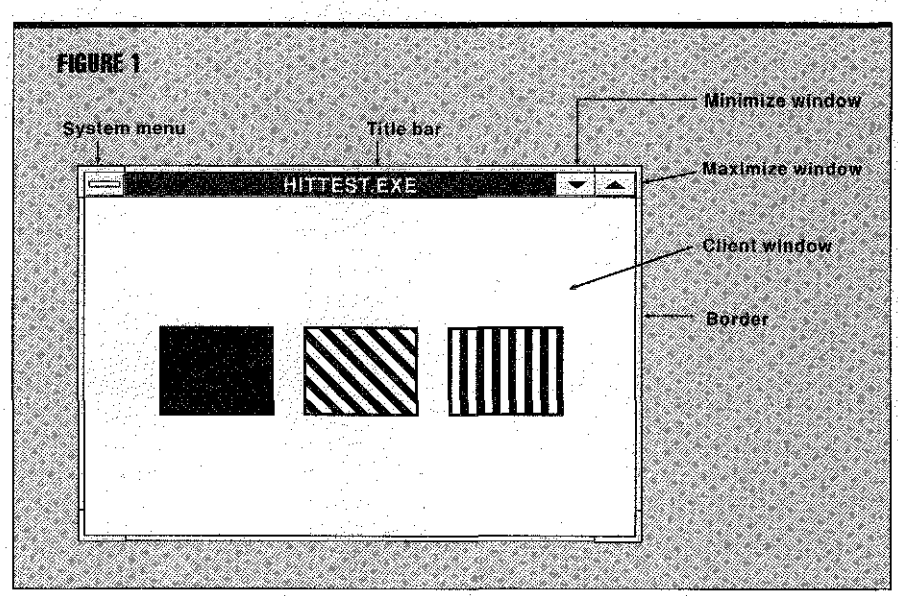
The first message a window procedure receives is WM_CREATE. All required initialization should be performed here. The only initialization

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

in **ClientWindowProc** is the creation of three child windows. As in **main**, the window procedure for a window class must be registered before the window can be created.

Notice the differences between this call to **WinRegisterClass** and the call in **main**. Obviously, the class name and the window procedure are different. Also, the last parameter is set to *sizeof(NPWDW_PARAMS)*. This specifies the size of the window data to be allocated to each window of this class. In this case, each window has a near pointer to a structure of type WDW_PARAMS associated with it. An additional class-style bit, CS_HITTEST, is set; this enables hit-testing for the window. The details of hit-testing will be discussed later.

Each child window is created with **WinCreateWindow**, which has several points of interest. The first parameter is set to *hwnd*; this indicates that the current window (*hwnd*) is the parent window of these child windows. The class name "Test" associates the child window with the window procedure registered in **WinRegisterClass**. WS_VISIBLE makes the window visible.



CHILD WINDOW PARAMETERS

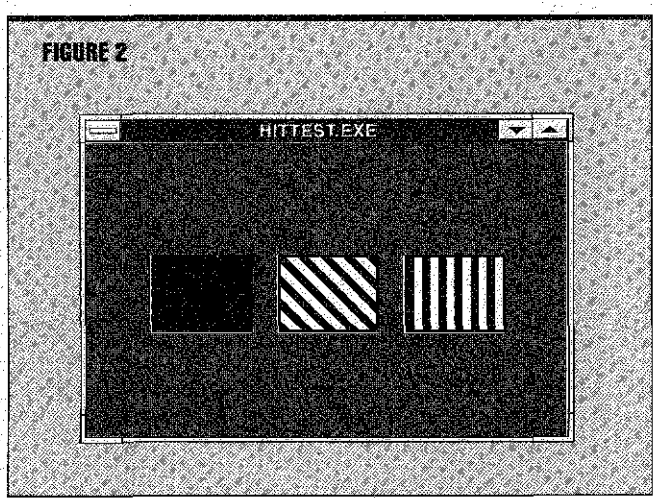
The next four parameters are the location and size of the window relative to the parent window. Since the width and height are both 0, the window is not displayed. The next parameter sets the owner of the window to *hwnd*. This ensures that when *hwnd* dies, the children also die. The HWND_TOP parameter specifies the relative placement of sibling windows (which are irrelevant to this application).

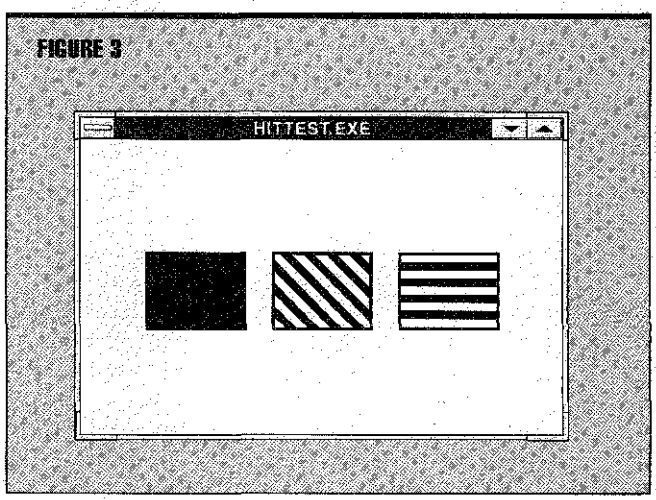
The parameter *i* illustrates another interesting point about windows: A window is identified by a unique handle and an ID. In simple applications like this, it is not necessary to remember the handle of a child window, because the handle can be obtained from the window ID. The parameter *vdata* is information that is specific to an individual window. Utilization of this data will be demonstrated later.

All the child windows are displayed during the WM_SIZE message. As mentioned earlier, the window handle is obtained from the window ID via a call to **WinWindowFromID**. Both the width and the height of the client window are passed in *mp2*. The child windows are sized at one-fifth the width and the height of the client window. The window position is determined by the formulas inside **WinSetWindowPos**. SWP_MOVE and SWP_SIZE indicate that the window's position and size change.

WINDOW UPDATING

WM_PAINT messages occur when a portion of the window is invalid and needs to be updated. All window updating should be done in WM_PAINT. This instance repaints the entire client window, as determined by **WinQueryWindowRect**. The effect of the CS_CLIPCHILDREN bit becomes obvious here: When the client window is repainted, the child windows are not. If CS_CLIPHILDREN were not set, WM_PAINT messages would be sent to **HittestWindowProc**, which would force the child windows to be





PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

If button 1 is double-clicked while positioned over the client window, the window is repainted only once.

repainted also. This is not much of a problem for this application, but it does cause an unsightly (and unnecessary) flicker.

A WM_BUTTON1DOWN message is received when mouse button 1 is clicked over a window. In **ClientWindowProc**, the color of the background is toggled and the window is invalidated.

The color change needs to occur right here and not in WM_PAINT, because the client window should change color only in response to a WM_BUTTON1DOWN message.

For example, consider a white client window that is moved partially off the screen. When the window is moved again, the portion that was off the screen becomes invalid and must be redrawn. If the color were changed every time a WM_PAINT message was received, this new region would be painted pink.

INVALID AREAS

WinInvalidateRect declares a region of the specified window to be invalid and sends a WM_PAINT message to force the invalid area to be repainted. In this case, the window is *hwnd*, or the client window. The second argument indicates the invalid area of the window.

Pay attention to the relationship between the invalid rectangle as defined by **WinInvalidateRect** and the area to be repainted as determined by **WinQueryWindowRect**. "Null" indicates that the whole window is invalid. If an actual RECTL structure were passed instead, only the area delineated by the structure elements would be repainted while processing a WM_PAINT message.

This is important: Only invalid areas are redrawn. Conversely, if the **WinFillRect** call in WM_PAINT specified a small rectangle, only the specified area would be painted.

If the entire client window is not painted during a WM_PAINT message, the screen beneath the client window remains visible. For an interesting insight into how OS/2 remembers window appearance, try substituting different coordinates into **WinFillRect** and moving the window around the screen.

The third parameter is set to FALSE, indicating that child windows are not to be updated. An interesting relationship exists between this parameter and the CS_CLIPCHILDREN bit. If the parameter is true, all the child windows are redrawn, effectively overriding the CS_CLIPCHILDREN setting. If CS_CLIPCHILDREN is not set, the child windows are updated regardless of this value.

Pressing a mouse button twice within a short time is called "double-clicking." This application provides a good demonstration of what happens when a double click occurs:

If button 1 is double-clicked while positioned over the client window, the window is repainted only once, because a double click sends two messages: WM_BUTTON1DBLCLK and WM_BUTTON1DOWN. The WM_BUTTON1DBLCLK message is then passed to the default window procedure (which does nothing), and WM_BUTTON1DOWN processes normally.

THREE CHILD WINDOWS

HittestWindowProc is the window procedure for the three child windows. The hit-testing logic is located here. Notice the call to **WinQueryWindowUShort** before the switch statement. This establishes addressability to the widow-specific data established in the **WinRegisterClass** call in **ClientWindowProc**. For windows of class "Hit-

LISTING 1

```
#
#   Hittest.exe make file
#

all:    hittest.exe rc.exe

hittest.exe : hittest.obj hittest.def hittest
     link /co /nod hittest, /align:16, NUL, slibcp os2, hittest

rc.exe : rc.obj rc.def hittest
     link /co /nod rc, /align:16, NUL, slibcp os2, rc


hittest.obj : hittest.c hittest
     cl /Zi /c /G2sw /W3 hittest.c

rc.obj : rc.c hittest
     cl /Zi /c /G2sw /W3 rc.c
```

LISTING 2

```
#define INCL_WIN
#define INCL_GPI
#include <os2.h>

// typedefs

typedef  struct wdw_params {
    MRESULT response;
    LONG    pattern,
            pattern1,
            pattern2;
} WDW_PARAMS;
typedef WDW_PARAMS  *NPWDW_PARAMS;
typedef WDW_PARAMS FAR  *PWDW_PARAMS;


// global variables

HAB     vhab;                                   // anchor block handle
HHEAP   vhHeap;                                 // heap handle

WDW_PARAMS    vdata[] = {
    MRFROMSHORT(HT_DISCARD),     PATSYM_SOLID, PATSYM_SOLID, PATSYM_NOSHADE,
    MRFROMSHORT(HT_TRANSPARENT), PATSYM_DIAG1, PATSYM_DIAG1, PATSYM_DIAG3,
    MRFROMSHORT(HT_NORMAL),      PATSYM_VERT,  PATSYM_VERT,  PATSYM_HORIZ
};


// function declarations

MRESULT EXPENTRY    ClientWndProc(HWND, USHORT, MPARAM, MPARAM);
MRESULT EXPENTRY    HittestWndProc(HWND, USHORT, MPARAM, MPARAM);
VOID    main(VOID);
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

test," a near pointer is the only data **being stored. WinQueryWindowPtr** and **WinQueryWindowULong** can be used for other data types.

The per-window memory is allocated while processing the WM_CREATE message. **WinAllocMem** obtains memory from the heap created by the **WinCreateHeap** call in **main**. The data that was passed during **WinCreateWindow** is copied, and the address of the data structure is written into the first word of window-specific storage.

The WM_PAINT logic is the same as in **ClientWindowProc**. Instead of being painted a specified color, a window is filled with a pattern. The pattern to be used was passed to **HittestWindowProc** with the WM_CREATE message. This pattern is loaded with **GpiSetPattern**. **WinQueryWindowRect** is used to obtain the coordinates of the window. **GpiMove** and **GpiBox** are called to create and fill a box the size of the window.

Notice that the dimensions in *rcl* are decremented before they are passed to **GpiBox**. The fill mode is given as DRO_OUTLINEFILL. This forces **GpiBox** to fill the box with the current pattern and draw a border around the box. If the

A CLOSE LOOK AT MOUSE MESSAGES

Presentation Manager mouse messages are not posted to the application's message queue. Instead, they are stored in the system message queue. Like other messages, they are retrieved using **WinGetMsg** or **WinPeekMsg** calls. A window procedure processes a mouse message as it would any other message. There are five mouse messages:

- WM_MOUSEMOVE is sent every time the mouse is moved within a window.
- WM_BUTTON*DOWN is sent when the appropriate button on the mouse is pressed.
- WM_BUTTON*UP is sent when the appropriate mouse button is released.
- WM_BUTTON*DBLCLK is sent when a button is pressed twice in rapid succession.
- WM_HITTEST is sent under certain circumstances before every mouse message.

All PM messages have two message-specific parameters: *mp1* and *mp2*. These parameters are used to convey information specific to an individual message. Parameters *mp1* and *mp2* hold the same information for each of the first four messages. *mp1* is the position of the pointer at the time of the event. The position coordinates are relative to the window that actually receives the message.

Note how this relates to the example application. When button 1 is clicked over the center child window, the WM_BUTTON1DOWN message is redirected to **ClientWndProc**. The WM_BUTTON1DOWN message that is received by **ClientWndProc** contains coordinates relative to the client window, even though the mouse was actually positioned over a child window when the button was pressed—the coordinates are automatically translated by the system. When the mouse is clicked while over the right child window, the subsequent WM_BUTTON1DOWN message is sent to **HittestWndProc**. The associated coordinates are relative to the child window. Parameter *mp2* is the hit-test result. This is the response to a previous WM_HITTEST message.

A WM_BUTTON*DBLCLK message is always preceded by a WM_BUTTON*DOWN message. If a window is created with class-style CS_HITTEST, a WM_HITTEST message is sent before every mouse message. The response to this message determines the routing of the subsequent mouse-event message and the value of the second message parameter:

- HT_NORMAL causes the mouse message to be sent to the current window procedure.
- HT_TRANSPARENT causes the mouse message to be sent to the window procedure of the underlying window.
- HT_DISCARD causes the message to be discarded by the system; it is not sent anywhere.
- HT_ERROR is the same as HT_DISCARD, except that the machine is supposed to beep.

An application can also define its own responses to a WM_HITTEST message. These return values are passed to the appropriate window procedure as the second parameter of the redirected mouse-event message.

For example, assume window A and window B are sibling windows, with window A atop window B. Window A was created with a class-style CS_HITTEST; window B, without. When window A receives a WM_HITTEST message, it returns a value of 15. This has the same effect as HT_TRANSPARENT, except that when window B receives the redirected mouse message, *mp2* is set to 15 instead of 0. If window B is also created with class-style CS_HITTEST, the WM_HITTEST message received by window B has *mp2* set to 15. If the response to the WM_HITTEST message is HT_NORMAL, window B then receives a mouse message with *mp2* set to 0.

Note that the example application demonstrates an anomaly in hit-test processing. In the application, the center window passes all mouse messages to the underlying window by returning HT_TRANSPARENT when a WM_HITTEST message is received. If the return value is changed to 15, for instance, the mouse message is no longer routed to the underlying window, **ClientWndProc**. Instead, it is directed to **HittestWndProc**, as if the return value were HT_NORMAL. The second parameter of the message is set to 15, as returned by the previous WM_HITTEST message.

Similarly, when **ClientWndProc** receives a message rerouted from **HittestWndProc** using an HT_TRANSPARENT return code, *mp2* is always set to 0. It is therefore impossible for **ClientWndProc** to recognize mouse messages that have been redirected from another window.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
VOID main(VOID)
{

    static ULONG    FrameFlags;                     // frame flags
    HMQ     hmq;                                    // msg queue handle
    HWND    hwndFrame,                              // frame window handle
            hwndClient;                             // client window handle
    QMSG    qmsg;                                   // window message queue

    vhab = WinInitialize(0);
    hmq = WinCreateMsgQueue(vhab, 0);
    vhHeap = WinCreateHeap(0, 0, 0, 0, 0, 0);

    WinRegisterClass(vhab, "Hittest", ClientWndProc, CS_SIZEREDRAW, 0);

    FrameFlags = FCF_TITLEBAR | FCF_SYSMENU | FCF_SIZEBORDER | FCF_MINMAX |
                 FCF_SHELLPOSITION | FCF_TASKLIST;

    hwndFrame = WinCreateStdWindow(HWND_DESKTOP, WS_VISIBLE, &FrameFlags,
                                   "Hittest", NULL, WS_CLIPCHILDREN, NULL,
                                   NULL, &hwndClient);

    while (WinGetMsg(vhab, &qmsg, NULL, 0, 0))
        WinDispatchMsg(vhab, &qmsg);

    WinDestroyHeap(vhHeap);
    WinDestroyWindow(hwndFrame);
    WinDestroyMsgQueue(hmq);
    WinTerminate(vhab);

}


MRESULT EXPENTRY ClientWndProc(HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2)
{

    static LONG color;                              // background color
    HPS     hps;                                    // PS handle
    RECTL   rcl;                                    // window coordinates
    SHORT   i,                                      // window ID
            cxClient,                               // client window width
            cyClient,                               // & height
            wdwWidth,                               // child window width
            wdwHeight;                              // & height

    switch (msg) {
    case WM_CREATE :
        WinRegisterClass(vhab, "Test", HittestWndProc,
                         CS_SIZEREDRAW | CS_HITTEST,
                         sizeof(NPWDW_PARAMS));

        for (i = 0; i < 3; i++)
            WinCreateWindow(hwnd, "Test", NULL, WS_VISIBLE,
                            0, 0, 0, 0,
                            hwnd, HWND_TOP, i, (PVOID)&vdata[i], NULL);

        color = CLR_BACKGROUND;                     // initial bkg color
        break;

    case WM_SIZE:
        cxClient = SHORT1FROMMP(mp2);               // get size of client
        cyClient = SHORT2FROMMP(mp2);               // window

        wdwWidth = cxClient / 5;                    // size for child windows
        wdwHeight = cyClient / 5;

        for (i = 0; i < 3; i++)
            WinSetWindowPos(WinWindowFromID(hwnd, i), NULL,
                            ((i + 1) * (cxClient / 4)) - (wdwWidth / 2),
                            (cyClient / 2) - (wdwHeight / 2),
                            wdwWidth, wdwHeight,
                            SWP_MOVE | SWP_SIZE);

        break;

    case WM_PAINT:
        hps = WinBeginPaint(hwnd, NULL, NULL);
        WinQueryWindowRect(hwnd, &rcl);
        WinFillRect(hps, &rcl, color);
```

continued

"To the rescue... the C Toolbox offer a number of solid utilities to reformat, analyze and validate source code..."
---Bill Rubin, The C Gazette, Winter 1988

The C Programmer's Toolbox contains more than 20 tools for the serious C programmer. The tools are available in three different development environments: PC/MS DOS; Macintosh's MPW; and Sun's UNIX. Some of the major tools include:

CDecl Translates C declaration statements to/from English *(helps you decipher those complicated C variable declarations and construct new ones). {MPW & Sun only}*

CFlow™ Determines how a program is organized and what is in the runtime library *(quickly shows you function hierarchy and interdependancies)*

CHilite™ Highlights and prints your C source code *(great for creating your final documentation). {MPW only}*

CLint™ Syntactically checks one or more of your C source code files *(use this to find those hard to find, syntactically correct programming mistakes)*

Cpp Preprocesses C source code files that have ANSI macro definitions *(helps you port ANSI C code to/from your target environment). {MPW & Sun only}*

CPrint™ Reformats/beautifies your C source code *(makes code easy to read and ends disputes on source code formats).*

CritPath™ Determine a program's critical path. *(helps you improve your program's performance) {PC DOS only}*

CXref™ Cross references and checks usage of up to 17 different classes of symbols in your C source code *(cross references any size program and number of listings)*

PMon™ Monitors a program's execution *(non-invasively monitors your program's function execution, system service and BIOS usage). {PC DOS only}*

"... The tools I tried operated both quickly and flawlessly..."
---M. Steven Baker, Programmer's Journal, May 1988

C™ Programmer's Toolbox / MPW PC Sun

The tools include support for ANSI C, Microsoft C, Turbo C, Quick C, Sun UNIX C, Macintosh's MPW C, THINK C and more. The Toolbox easily handles any size program through its virtual memory system. Several tools also include a C macro processor that accepts K&R, ANSI and other C dialects.

C Programmer's Toolbox™
MPW ($295), PC Volume I or II ($99.95),
PC Volume I & II ($175), Sun ($495)
C Programmer's McTool Series™ (Macintosh - non-MPW)
McCPrint™ ($59.95), McCLint ($99.95)

"... it's a complete set of utility programs which can make the life of a C programmer easier... I really like this package..."
-- Scott Robert Ladd, Micro Cornucopia, November 1988

Call and Order today...
(CA residents, please add sales tax; Overseas shipping extra)

MasterCard VISA®

MMC AD Systems™
Box 360845
Milpitas, California 95035
(408) 263-0781

Number **94** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

top and right coordinates aren't decremented, only the bottom and left borders will be drawn.

WM_BUTTONIDOWN processing is identical to that in **ClientWindowProc,** with one exception. Since the procedure governs three windows, the previous pattern must be remembered on a per-window basis. This means that it is stored in the window data area and not in a static variable as in **ClientWindowProc**. Notice that the logic here is identical for all windows. If a WM_BUTTONIDOWN message is received, the window will change appearance.

MESSAGE DISPOSITION

WM_HITTEST controls the disposition of mouse messages. This capability is enabled when the CS_HITTEST bit is set during the **WinRegisterClass** call in **ClientWindowProc**. When hit-testing is enabled, every mouse message to a window procedure is preceded by a WM_HITTEST message.

Each of the three child windows has a predefined response to the WM_HITTEST message. This response was passed during WM_CREATE and is stored in the window data area. The array *vdata[]* indicates that each window has a different response: HT_DISCARD, HT_TRANSPARENT, or HT_NORMAL.

HT_DISCARD directs OS/2 to discard the mouse message. This is why the child window on the left never changes—it never gets a WM_BUTTONIDOWN message. HT_TRANSPARENT causes OS/2 to reroute the mouse message to the window below. Therefore, clicking on the center window causes the client window below to change color.

Again, **HittestWindowProc** never receives a WM_BUTTONIDOWN message, but this time **ClientWindowProc** does. The third window responds to WM_HITTEST with HT_NORMAL. The WM_BUTTONIDOWN messages are then sent to **HittestWindowProc**, which causes the window to change appearance.

A WM_DESTROY message informs the window procedure that a window is being destroyed. In **HittestWindowProc**, this means that the memory allocated in WM_CREATE must be freed. This is done with a call to **WinFreeMem**. Contrast this with **ClientWindowProc**, where no cleanup is required.

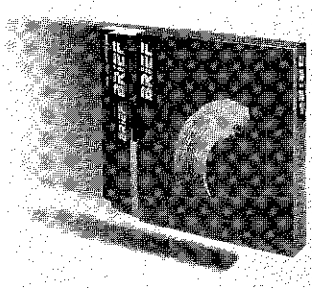
Your operating system finally caught up with our editor.

Introducing BRIEF for OS/2.
It's the most versatile editor available, in a powerful new environment.

Now, you can have the best multi-level undo available, potent search and translate capabilities, and two powerful (but easy-to-use) macro languages. Plus all the inherent advantages of OS/2, like multitasking.

And BRIEF is still easy to use, and easy to learn. Commands are consistent and intuitive, and the documentation is clear and concise. Best of all, it's still affordable—just $199.

If you've made the investment in time and money to move up to OS/2, make sure you're working with an editor as advanced as your operating system.



Call today to order your copy of BRIEF for OS/2. And stop playing catch-up.

Special discount for DOS BRIEF users.

Call 1-800-343-7308.

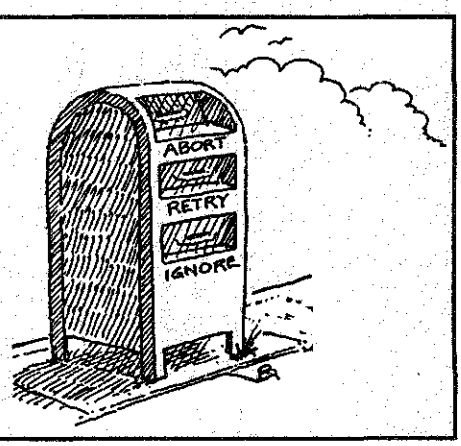
BRIEF® for OS/2

UnderWare, Inc., 321 Columbus Ave., Boston, MA 02116 (617) 267-9743

Number 309 on the Reader Service Card

HOW'RE WE DOING?

Please let us know what you think of this issue—fill out the peer-review card facing page 88, and drop it in the mail. (The postage is on us.)



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        WinEndPaint(hps);
        break;

    case WM_BUTTON1DOWN:
        color = (color == CLR_BACKGROUND) ? CLR_PINK : CLR_BACKGROUND;
        WinInvalidateRect(hwnd, NULL, FALSE);
        break;

    default:
        return(WinDefWindowProc(hwnd, msg, mp1, mp2));
    }

    return(NULL);

}


MRESULT EXPENTRY HittestWndProc(HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2)
{

    HPS     hps;                                    // PS handle
    NPWDW_PARAMS WinData;                           // window specific data
    POINTL  ptl;                                    // point coordinates
    RECTL   rcl;                                    // rectangle coordinates

    WinData = (NPWDW_PARAMS)WinQueryWindowUShort(hwnd, 0);

    switch (msg) {
    case WM_CREATE:
        WinData = (NPWDW_PARAMS)WinAllocMem(vhHeap, sizeof(WDW_PARAMS));

        WinData->pattern = ((PWDW_PARAMS)mp1)->pattern1;
        WinData->pattern1 = ((PWDW_PARAMS)mp1)->pattern1;
        WinData->pattern2 = ((PWDW_PARAMS)mp1)->pattern2;
        WinData->response = ((PWDW_PARAMS)mp1)->response;

        WinSetWindowUShort(hwnd, 0, (USHORT)WinData);
        break;

    case WM_PAINT:
        hps = WinBeginPaint(hwnd, NULL, NULL);
        WinQueryWindowRect(hwnd, &rcl);

        GpiErase(hps);

        GpiSetPattern(hps, WinData->pattern);
        ptl.x = 0;                                  // lower left corner
        ptl.y = 0;                                  // of window
        GpiMove(hps, &ptl);
        ptl.x = --rcl.xRight;                       // decrement so we can
        ptl.y = --rcl.yTop;                         // see the border
        GpiBox(hps, DRO_OUTLINEFILL, &ptl, 0L, 0L); // draw box w/border

        WinEndPaint(hps);
        break;

    case WM_BUTTON1DOWN:
        WinData->pattern = (WinData->pattern == WinData->pattern1) ?
                                WinData->pattern2 : WinData->pattern1;
        WinInvalidateRect(hwnd, NULL, FALSE);
        break;

    case WM_HITTEST:
        return(WinData->response);

    case WM_DESTROY:
        WinFreeMem(vhHeap, (NPBYTE)WinData, sizeof(WDW_PARAMS));
        break;

    default:
        return(WinDefWindowProc(hwnd, msg, mp1, mp2));
    }

    return(NULL);

}
```

LISTING 3

```
NAME        HITTEST WINDOWAPI

DESCRIPTION 'Hittest demonstration program'
PROTMODE
HEAPSIZE    1024
STACKSIZE   8192
EXPORTS     ClientWndProc
            HittestWndProc
```


NEW

Develop Powerful Image-based Applications Quickly and Easily

Introducing VICTOR, video capture and image processing library

VICTOR simplifies development of scientific imaging, quality control, security, and image database software. Victor gives you complete control of your video frame grabber and also includes image processing, display, and TIFF/PCX file handling routines.

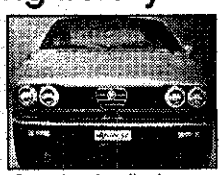
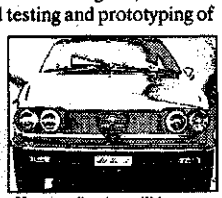


Image-based applications can be developed in QuickC and Turbo C environments.

Application development is simplified because we've taken care of the details of device control and image processing. All the hard low level coding has been done -- you don't have to worry about bit twiddling, video timing, etc. This means you can concentrate on your application. Your software can have features such as: live video on VGA, zoom, display multiple images with text and graphics, pixel editing and image processing. And, to get you up and running fast, we've included Zip Image Processing software for rapid testing and prototyping of image processing and display functions.

Victor supports Microsoft C, QuickC, and TurboC, includes over 100 functions, demonstration and prototyping software, full documentation, and source code for device control routines... all for only $195. Full source code add $100, video frame grabbers available.



Your application will have powerful image handling capabilities.

Call (314) 962-7833 to order
VISA/MC/COD

Catenary Systems 470 Belleview St Louis MO 63119 (314) 9627833

Number 321 on the Reader Service Card

Peek, Poke

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Exploring extended-memory block moves

Accessing high memory

By Larry Smith

Several months ago, in *Programmer's Journal* Volume 6.6 (November/December 1988), Jake Richter described how to access the first 64K above 1MB on an 80*x*86 machine. In a letter to *Programmer's Journal* (Volume 7.4, July/August 1989), I mentioned that I had already implemented several programs using this idea. Here I get to explain the idea and what one can do with it.

The 8086 has only 20 address lines; clearly, the largest address that the processor can construct is 2^{20}, or 1MB. The 80286 has 24 address lines, and—lo and behold—the 80386 has 32. That means that an address constructed from

```
DS = 0FFFFH
SI = 1234H
```

ought to point somewhere outside of the normal 1MB address space associated with the Intel processors as some kind of unbreakable boundary in real mode. True—but, in the interest of backward compatibility with who knows what program from MicroTrash, IBM decided to *prevent* the 80286 from accessing this memory location. They built a line into the motherboard of the original AT (since copied by all clone manufacturers) to wrap address line 20 back to address line 0, effectively defeating the ability of the processor to access a bit more memory in real mode. This "wrap" had to be software switchable to allow programs to access memory above 1MB.

A routine in the POST, called GATE_A20, toggles the switch open for the POST and toggles it closed again at the close of the check for extended memory. My reading of the IBM ROM BIOS for the AT shows that only one other routine—the BIOS extended-memory block-move functions—accesses the GATE_A20 routine. Since it would certainly "violate the rules of DOS" for a program to access extended memory any other way than through the BIOS (Isn't that a condemnation of Microsoft's code?), this opens the way for the programmer to undo what IBM did.

For consistency of terminology in this article, let's refer to memory in the range 0FFFFh:0000h–0FFFFh:0FFFFh as "hi-memory." All we have to do to access hi-memory is write an interrupt handler to intercept calls to the extended-memory block-move functions and toggle the address line 20 open after each such call. That makes hi-memory available in real mode,

Larry Smith is a programmer living in Friedland, West Germany. Readers may contact Mr. Smith via Bitnet; his ID is LSMITH@DGOGWDG1.

FIGURE 1

```
                ORG 0
;----------------
;DEVICE HEADER - must be at offset zero within device driver
;----------------
          DD      -1              ;becomes pointer to next device header
          DW      0800H           ;attribute (IBM format block device)
                                  ;supports OPEN/CLOSE/RM calls
          DW      STRATEGY        ;pointer to device "strategy" routine
          DW      INTERRUPT       ;pointer to device "interrupt handler"
          DB      1               ;number of block devices
          DB      7 DUP (?)       ;7 byte filler (remainder of 8-byte
                                  ;name)

;----------------
;This volume label is placed into the directory of the new VDISK.
;It is also used by VDISK386 to determine if a VDISK386 driver has already
;been loaded. Its placement in the source is INVIOLATE!
;----------------
VOL_LABEL  DB   'VDISK 3.0  '   ;00-10 volume name 11 BYTES LONG
           DB     28H           ;11-11 attribute (volume label)
           DT     0             ;12-21 reserved
           DW     6000H         ;22-23 time=12:00 noon
           DW     0986H         ;24-25 date=12/06/84
VOL_LABEL_LEN  EQU  $-VOL_LABEL ;length of volume label

;----------------
;The following field, in the first extended-memory VDISK device driver,
;is the 24-bit address of the first free byte of extended memory.
;This address is not in the common offset/segment format.
;The initial value, 10 0000H, is 1 megabyte.
;N.B. THE OFFSET OF THIS FIELD IS UNCHANGEABLE!
;               ORG 2CH
;----------------
AVAILABLE_EXT_MEMORY:
AVAIL_LO   DW   0               ;address of first free byte of
AVAIL_HI   DB   EM_START        ;extended memory
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

To access hi-memory, write an interrupt handler to intercept calls to the extended-memory block-move functions and open address line 20 after each call.

giving us 1MB plus 64K minus the first 16 bytes in the segment beginning at 0FFFFh to play with. We are missing the last 16 bytes of the segment beginning at 0FFFFh; these 16 bytes are in ROM and so are of no use anyway.

So much for theory—how about practice? I have chosen to use this newfound memory for most of my terminate-and-stay-resident programs (TSRs). I say "most" because a number of cautions are connected with using this memory.

The primary caution is not to point interrupts that are driven by the timer into this newfound memory. Such an interrupt handler might be called while the address line 20 is gated closed; therefore the routine called will be wrapped into segment 0, where who knows what address will be called. Another restriction concerns interrupt handlers driven by the keyboard, for the same reason.

Of course you could always split such routines into a portion in normal memory for the interrupt handler and a portion in hi-memory for the code called by the interrupt handler. Let your imagination fill in the details.

A second caution concerns disk I/O. DOS insists on using normalized pointers—that is, putting as much of a memory address into the segment part as possible. DOS does not know about the 32-bit registers on the 80386; so, in constructing a normalized pointer from

```
DS = 0FFFFH
DX = 01234H
```

DOS arrives at the following address:

```
DS = 0000H
DX = 1224H
```

Too bad. A call to DOS function 3Fh to read a disk file into the memory buffer pointed to by DS:[DX] = 0FFFFh:1234h will be read into the memory location starting at 0000h:1233h, with unexpected consequences!

A third caution concerns other programs (VDISK in particular) that make use of extended memory: How are they to be prevented from using the newfound segment (almost) at

LISTING 1

```
;HI-MEM.ASM
;This program contains a BIOS patch in a TSR running in HI-MEMORY.
;The module contains:
;       GATE_A20:This module is packed into the PSP before termination. It
;resides in normal memory as a TSR, traps calls to the BIOS extended memory
;block move function (INT 15H : AH = 87H) and gates address line 20 open
;after each call.
;
;       EUROTRS:Printer interface to the TSR 2100 printer.
;

TRUE    equ 0ffh
FALSE   equ 0

 80386  equ TRUE
TESTING equ FALSE

        INCLUDE C:\MASM\LIB\STD_MACS.ASM

ON      equ 0ffh                ;Uniform codes for ON/OFF toggles
OFF     equ 00h

LF      EQU     0AH             ;ASCII LINEFEED
CR      EQU     0DH             ;ASCII RETURN

;equates for the INT 2fH handler
KERNEL_ID       equ 0FFh
EUROTRS_ID      equ KERNEL_ID
INSTALLED       equ 0FFh        ;indicates installed

;function codes supported: ( placed by caller in AL )
GET_INSTALLED_STATE     equ 0
;This function is supported by EUROTRS
STATUS                  equ 4           ;return ES:[DI] ---> STATUS_FLAG

;8042 KEYBOARD CONTROLLER equates

KB_STATUS_PORT  equ 64H
PORT_A          equ 60H

ENABLE_A20      equ 0DFH
DISABLE_A20     equ 0DDH
WRITE2_8042     equ 0D1H

SUCCESS         equ 0   ;RETURN CODES
FAILURE         equ 2   ;8042 could not perform the command

BUFFER_FULL     equ FAILURE

CALLF   macro
        db 09ah         ;code for a far call to the immediate following
                        ;dword pointer
        endm

;
;ACHTUNG!!! BE VERY CAREFUL WITH ASSUME STATEMENTS THROUGHOUT!!!!
;EACH PROCEDURE (AND I MEAN EACH!) SHOULD HAVE AN ASSUME STATEMENT AS PART
;OF ITS HEADER. THINK OF IT AS PART OF THE PROCEDURE DECLARATION.
;

TSRGROUP        GROUP CODESEG,DATASEG,INITSEG

                TSR     equ TSRGROUP    ;for group overrides


;SPECIFY THE LOAD ORDER FOR THE LINKER
CODESEG SEGMENT BYTE PUBLIC 'PROG'
CODESEG ENDS
;
DATASEG SEGMENT BYTE PUBLIC 'DATA'
DATASEG ENDS
;
INITSEG SEGMENT BYTE PUBLIC 'INIT'
INITSEG ENDS
;

;SOME USEFUL ADDRESSES IN PSP
CODESEG SEGMENT BYTE PUBLIC 'PROG'
        ORG 2CH
ENVIRONMENT     dw ?
CODESEG ENDS


;START OF ACTUAL PROGRAM SEGMENTS

DATASEG SEGMENT BYTE PUBLIC 'DATA'
                ASSUME DS:TSRGROUP

;EUROTRS resident data area
STATUS_FLAG     db ON

CONVERSION_TABLE        LABEL BYTE
;This is a simple look-up table to convert IBM hi ASCII codes into codes
;understood by my not-so-compatible printer.
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

0FFFFh? There seem to be two accepted (even somewhat documented) ways for DOS programs to inform themselves about the availability of extended memory. The first of these is what I call the VDISK kludge. When VDISK loads, it captures INT 19h (the interrupt used to start a warm boot) and points it at a simple routine of its own.

The absolute beginning of the VDISK code is shown in Figure 1. The address at AVAILABLE_EXT_MEMORY indicates the start of available extended memory; the first copy of VDISK will place its buffer beginning at this address. Succeeding copies of VDISK (and other programs using this method) will get the address of INT 19h, which now points to VDISK, and check the 20-bit address at offset 2Ch in the VDISK segment for the beginning of available extended memory. All we need to do is change the byte at offset 2Eh to 11h from the current value of 10h. Since IBM graciously provides the source code for VDISK (at least for DOS 3.10), this is an easy matter. If you don't have the source code, you can make the change using a debugger.

But what if you don't use VDISK? If neither another program nor a device driver is using extended memory, you need do nothing. If none of your device drivers is using extended memory but another program is, then you could put an INT 19h driver of the form

```
JMP CS:[OLD_INT_19H]
```

into the hi-memory code and put the 24-bit address 11 0000h at offset 2Ch in the program segment prefix (PSP). If you have device drivers that use extended memory, then you will have to write a character device driver (call it NUL so that DOS won't find it), capture INT 19h and point it at the driver's INT 19h jump instruction, and place the magic number 11 0000h at offset 2C in the driver.

The second way that programs inform themselves about extended memory is by using extended memory from the top down, trapping INT 15h function 88h (extended-memory size) to adjust the top of extended memory to reflect their usage of it. All the programs and devices I have seen that use this method also use the VDISK kludge; some version of the VDISK kludge ought to suffice.

Listing 1 is a printer-driver program that makes use of these techniques. (My Tandy printer does not understand the IBM high-ASCII codes the way IBM intended.) To minimize the resident code, the INT 15h handler is moved into the PSP. The program is only a sample of the ideas involved, and I doubt that there are any astoundingly new ideas for you in the actual code. Still, you might find the paradigm useful, assuming that you use VDISK and that you have modified it as described.

All code in this issue is available through our bulletin board services or directly from *PJ* on disk. For more information see pages 8 and 88.

```
;This is a translation table for codes 128 to 255 ONLY.
        DB 80H,088H,0B8H,83H,0B6H,0A1H,86H,0A2H
        DB 88H,89H,08DH,0A9H,0AFH,0BAH,0B1H,8FH
D144    DB 90H,91H,92H,93H,0B7H,95H,96H,97H
        DB 98H,0B2H,0B3H,0B4H,0A3H,0B0H,9EH,0BFH
D160    DB 0A0H,0A1H,0A2H,0A3H,0A4H,0A5H,0A6H,0A7H
        DB 0A8H,0F0H,0F2H,0AEH,0ACH,0ADH,0AEH,0AFH
D176    DB 0EFH,0EFH,0EFH,0F5H,0F9H,0B5H,0B6H,0B7H
        DB 0B8H,0B9H,0BAH,0BBH,0BCH,0BDH,0BEH,0F2H
D192    DB 0F6H,0F8H,0F3H,0F4H,0F1H,0FAH,0C6H,0C7H
        DB 0C8H,0C9H,0CAH,0CBH,0CCH,0CDH,0CEH,0CFH
D208    DB 0D0H,0D1H,0D2H,0D3H,0D4H,0D5H,0D6H,0D7H
        DB 0D8H,0F7H,0F0H,0EFH,0E8H,0E9H,0EAH,0E7H
D224    DB 0E0H,0B9H,0E2H,0E3H,0E4H,0E5H,0A5H,0E7H
        DB 0E8H,0E9H,0EAH,0EBH,0ECH,0EDH,0EEH,0EFH
D240    DB 0F0H,0F1H,0F2H,0F3H,0F4H,0F5H,0F6H,0A6H
        DB 0A7H,0F9H,0FAH,0FBH,0FCH,0FDH,0FEH,020H

DATASEG ENDS


CODESEG SEGMENT BYTE PUBLIC 'PROG'
        ASSUME CS:TSRGROUP,DS:TSRGROUP ;ES:TSRGROUP,SS:TSRGROUP
        ORG 10H ;THE FIRST 16 BYTES OF THE SEGMENT @ OFFFFH ARE IN ROM

INT_17H label dword     ;BIOS Print routine
INT_17H_OFF     dw ?
INT_17H_SEG     dw ?

INT_2FH label dword     ;DOS multiplex interrupt
INT_2FH_OFF     dw ?
INT_2FH_SEG     dw ?

        ORG 5CH         ;BELOW THIS ADDRESS IS FOR DOS
INT_15H label dword     ;storage for the address of INT 15H to chain to
INT_15H_OFF     dw ?
INT_15H_SEG     dw ?

TARGET  label byte      ;target for the moved INT 15H handler

ZZZ     LABEL BYTE      ;N.B. THIS LABEL MUST BE AT AN OFFSET < 100H -
                        ;GATE_A20_SIZE
        PUBLIC  ZZZ

;RESIDENT CODE
                ORG 100H

BEGIN:          jmp SETUP
```

```
; MULTIPLEX  INTERRUPT

INT_2FH_HANDLER proc far
;Look for the ID of KERNEL and the individual modules.
        cmp ah,KERNEL_ID
        je MINE

        jmp CS:[INT_2FH]

MINE:   or al,al
        jnz INVALID_FUNCTION

        mov al,INSTALLED
INVALID_FUNCTION:
        iret

EUROTRS:
;       ENTRY:  AH = EUROTRS_ID
        cmp al,STATUS           ;only STATUS and RETURN_INSTALLED_STATE
        jne MINE                ;are supported by EUROTRS

        push CS
        pop ES                  ;return ES:[DI] ---> STATUS_FLAG
        mov di,offset TSR:[STATUS_FLAG]
        iret

INT_2FH_HANDLER endp

;       EUROTRS

INT_17H_HANDLER proc far
;Interface to the TRS 2100 printer.

                ASSUME DS:NOTHING,ES:NOTHING

        cmp CS:[STATUS_FLAG],ON
        jne DONE

        cmp ah,0                        ;PRINTER OUTPUT?
        jne DONE                        ;NO

        test al,80H                     ;HI ASCII CODE?
        jz DONE                         ;NO

        and al,7FH                      ;YES, CLOBBER HI BIT AND TRANSLATE
        push bx                         ;SAVE REGISTER
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        mov bx,offset TSR:[CONVERSION_TABLE]    ;CS:[BX] ---> CONVERSION
TABLE
        xlat byte ptr CS:[bx]           ;CONVERT
        pop bx

DONE:   jmp  CS:[INT_17H]               ;PASS ON THE TRANSLATED CODE

INT_17H_HANDLER endp

CODESEG ENDS


INITSEG SEGMENT BYTE PUBLIC 'INIT'
                ASSUME CS:TSRGROUP,DS:TSRGROUP
;Throwaway code and data for the initialization of the resident portion

;INITDATA
PROMPT          db 'KER386 installed. Bye!',CR,LF,'$'

LOADED_MSG      db 'KER386 is already loaded. Aborting!',CR,LF,'$'

BAD_DOS_MSG     db 'KER386 requires DOS 3.0 or higher. Aborting!',CR,LF,'$'

HARDERROR_MSG   db 'Cannot load KER386!',CR,LF,'$'
;INITCODE
SETUP:  mov ah,30h              ;be sure DOS version is ok
        int 21h
        cmp al,3
        jae VERSION_OK

        mov dx,offset TSR:[BAD_DOS_MSG]
        jmps ABORT_EXIT

VERSION_OK:
        mov ax,KERNEL_ID shl 8
        int 2Fh
        or al,al
        jz CONTINUE

ABORT:  mov dx,offset TSR:[LOADED_MSG]
ABORT_EXIT:
        mov ah,09h              ;DOS display ASCII string function
        int 21h
        int 20h

CONTINUE:
        call GATE_A20           ;open up the first 64 KB above 1 MEG
        or al,al
        jz INSTALL

        mov dx,offset TSR:[HARDERROR_MSG]
        jmps ABORT_EXIT

INSTALL:
;Move the essential functions to HI RAM
        mov si,10h
        mov di,si
        mov ax,0FFFFh
        mov ES,ax
        mov cx,offset TSR:[THE_END] - 10h
        CLD
        rep movsb

        CALLF                   ;call the routine at the address in the
                                ;next dword
        dw offset TSR:[HI_RAM_ENTRY]
        dw 0FFFFH
;Code in HI RAM ends with a RETF, so control returns here after call.
        mov ax,CS
        mov DS,ax       ;restore DS to CS
;get and save the address of the INT 15H routine
        mov ax,3515h    ;get the address of INT 15h from DOS
        int 21h
        mov [INT_15H_OFF],bx
        mov [INT_15H_SEG],ES

        mov ax,DS       ;restore ES to DS=CS
        mov ES,ax
;Move the code for gate_a20 into the PSP and establish residency.
        mov si,offset TSR:[INT_15H_HANDLER]
        mov di,offset TSR:[TARGET]
        mov cx,GATE_A20_SIZE
        CLD
        rep movsb
;now reset the interrupt routine
        mov ax,2515h
        mov dx,offset TSR:[TARGET]
        int 21h
;free up the environment segment
        mov ES,TSR:[ENVIRONMENT]
        mov ah,49h
        int 21h
;establish TSR status
        mov dx,0FFH
        int 27h

;       INIT ENTRY POINT IN HI MEMORY

HI_RAM_ENTRY:
;The entry point called by the original copy of the code
;after the move to high memory.
        mov ax,CS
        mov DS,ax
        mov ES,ax
;chain into INT 2Fh chain
        mov ax,352Fh
        int 21h
        mov [INT_2FH_SEG],ES
        mov [INT_2FH_OFF],bx
        mov ax,252Fh
        mov dx,offset TSR:[INT_2FH_HANDLER]
        int 21h
;now Bios PRINT INTERRUPT
        mov ax,3517h    ;get the address of INT 15h from DOS
        int 21h
        mov [INT_17H_OFF],bx
        mov [INT_17H_SEG],es
        mov ax,2517h
        mov dx,offset TSR:[INT_17H_HANDLER]
        int 21h

RETURN2CALLER:
        RETF                                    ;RETURN TO CODE IN NORMAL MEMORY

;The next label marks the end of code transferred to hi memory
THE_END label byte

;This code is packed into the PSP before termination.

INT_15H_HANDLER proc far
;Patch BIOS EXTENDED MEMORY BLOCK MOVE function to gate address line 20.
        cmp ah,87H              ;BIOS extended memory block move function
        je REGATE_A20

        jmp CS:[INT_15H]        ;N.B. must be "relocatable"
REGATE_A20:
        pushf                   ;simulate an interrupt
        call CS:[INT_15H]       ;must be "relocatable"
        pushf                   ;save the flags from the BIOS call
        call GATE_A20
        popf
        RETF 2          ;throw away user flags and return from BIOS call.

INT_15H_HANDLER endp

GATE_A20        proc near
;       ENTRY:  NONE
;       EXIT:   AL = STATUS CODE
        push ax
        CLI
        call EMPTY_8042         ;be sure 8042 input buffer is empty
        jnz GATE_A20_X          ;quit if no go

        mov al,WRITE2_8042      ;enable the 8042 for a command
        out KB_STATUS_PORT,al
        call EMPTY_8042
        jnz GATE_A20_X

        mov al,ENABLE_A20       ;get the command
        out PORT_A,al           ;send it to the 8042
        call EMPTY_8042
GATE_A20_X:
        pop ax
        RET
GATE_A20        endp

EMPTY_8042      proc near
;This routine waits for the 8042 input buffer to empty
;       ENTRY:  NONE (AX SAVED BY CALLER)
;       EXIT:   AL = RETURN CODE (SUCCESS or FAILURE)
        push cx                 ;save scratch register
        xor cx,cx               ;set up a time out counter
EMPTY_LOOP:
        in al,KB_STATUS_PORT
        and al,BUFFER_FULL
        loopnz EMPTY_LOOP
        pop cx
        RET
EMPTY_8042      endp

GATE_A20_SIZE   equ $ - INT_15H_HANDLER

INITSEG ENDS

        END BEGIN
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Evaluating the performance of C Compilers

Benchmarking—fact or fiction?

By Rick Naro

Recent events in Massachusetts should put compiler vendors on notice when it comes to making performance claims in print advertising. Recently, the state department of consumer affairs started an investigation into the advertising of the top two cola companies when each claimed more people preferred their product. Since it was obvious that two such claims must be mutually exclusive, the state launched the investigation to determine if any consumer-protection laws concerning false advertising were broken.

Picking up any of the popular computer magazines shows a similar climate in the C-compiler marketplace, and, with no action being taken on programmers' behalf, we are on our own to evaluate the validity of the claims.

Like choosing a cola, deciding which compiler performs better often comes down to making a personal judgment call. As it is nearly impossible to duplicate exactly the benchmark conditions employed in advertising, consistently reproducing the published results is out of the question. (And even if the claims are true as stated, are they relevant to the requirements of the typical application?) Nevertheless, we don't need to give up and simply ignore the conflicting claims—analyzing some compiler-performance claims from a personal perspective can shed some light on an otherwise dark subject.

As programmers, we are primarily concerned with the quality of our own code and the tools that form our daily working environment. The compiler, the jewel of a complete software-development package, is complemented by run-time libraries, a debugger, and other useful programming utilities, each of which is intended to enhance productivity. Because performance issues rarely rear their ugly heads until the very end of the software-development cycle, the availability of high-quality run-time libraries, debuggers, programming shells, and other tools is all the more reason to ignore specific benchmark claims and instead evaluate the worth of the entire software-development environment.

While general-purpose benchmarks have their place, the ultimate benchmark will always be your own application. This presents a minor dilemma in that unless you have a working crystal ball, it is difficult to predict the effect on performance by a compiler before the core algorithms and program structure have been developed. Being at the mercy of the advertised claims and independent compiler reviews is not so bad as it seems: Since most name-brand compilers are as good as they are inexpensive, costly mistakes become unlikely. Yet the seriousness of compiler vendors pushing a single benchmark as sufficient justification for purchasing a compiler gives us the opportunity to belittle the results and poke some fun at the expense of benchmarking claims in general.

Rick Naro is president of Paradigm Systems, a software-development firm specializing in embedded systems. He can be reached at P.O. Box 152, Milford, MA 01757, or on BIX as PARADIGM.

SYNTHETIC BENCHMARKS

The use of synthetic benchmarks such as the Sieve and Dhrystone in performance-analysis claims should immediately be suspect: Besides being trivially short and performing no useful work, their results are routinely stretched to achieve a desired objective. To demonstrate the inherent inexactness of synthetic benchmarks, we can start with the seemingly innocuous Sieve of Eratosthenes code, in Listing 1.

Used at one time or another by just about every compiler vendor or product reviewer, this benchmark is usually seen as a measure of general-purpose performance. Demonstrating the ease with which the Sieve benchmark is sensitive to manipulation, each of three popular C compilers can lay claim to the best overall performance. Such sensitivity to simple and unpublished methods is the reason that benchmark claims for synthetic applications should be treated skeptically, especially when they are incompletely documented.

THREE WINNERS

The mental preparations for this exercise are simple. For starters, you need to put yourself in the shoes of the product-marketing manager for a major compiler vendor preparing an ad campaign for an upcoming release. Now, as every marketing type knows, if at first you don't

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

One important consideration . . . is your own programming skills.

succeed, you need to reposition and try again. Since each benchmark run produces a single winner, less lucky vendors will have the opportunity to evaluate the initial results before making a minor tweak.

Compiling the baseline code provides our first set of results and our initial victor. For the record, all benchmarks were measured on a vanilla PC clone running at 10 MHz and using the Microsoft C 5.1, Turbo C 2.0, and WATCOM C 7.0 compilers. All benchmarks were built with as much optimization as possible using the following command line options:

WATCOM C:	*-d2 -ox*
Microsoft C:	*/Zi /Ox*
Turbo C:	*-v -G -O -Z*

All three compilers were built with debugging information enabled, followed by the options to reenable optimizations.

To achieve the accuracy needed to compare these compilers, the execution time of the function **sieve** was measured using the software-performance analyzer INSIDE! in the function-timing-analysis mode. The execution times shown in the tables in this article are the average of the ten calls made to **sieve** and do not include the time spent in **main** or the compiler start-up and shutdown code. The actual results for any instance of **sieve** varied by no more than 50 microseconds, making these execution times very accurate.

AND THE WINNERS ARE . . .

At the end of the first run (Table 1), vendor C is done and is already busy preparing the advertising declaring its product the winner by a 10-percent margin. Unfortunately, benchmarking is never so simple as it seems, and a closer look from the perspective of vendors A and B is required for fairness. An obvious starting point is the wide difference in execution time between the small- and large-memory-model results from compiler B.

An examination of the link map from the large-model build showed that compiler B put the array **flags[]** in a segment not addressable via the contents of the DS segment register. This introduces a considerable disadvantage, since each access to data allocated in the segment requires that ES be loaded before the access can take place. The simple addition of an initializer to the array of flags in line 10:

```
char            flags[SIZE+1] = { 0 } ;
```

forces the compiler to allocate the array in a segment addressable via the DS segment register. Since this insignificant change in no way affects the results of the prime number calculations, its use should be considered perfectly legitimate by all participants. Retesting each of the compilers by vendor B with this minor change installed generated the results shown in Table 2.

This time, compiler B achieved identical execution times for both the small- and the large-memory models, helping win the overall competition by a much faster implementation of the large-model benchmark than compiler C's.

In the first two trials, compiler A lagged behind the other two by what must seem an insurmountable margin. While it might seem difficult to improve compiler A's execution times by the 30 percent or more needed to surpass compilers B and C, keep in mind that nothing should be out of the grasp of the astute benchmarking specialist. Once again, we must return to the original source code and make a minor change, this time to the code that initializes the array.

Using INSIDE! in the source-line-timing mode quickly pinpoints the array initialization as the bottleneck. The only adjustment is the replacement of the array-initialization loop in lines 37 and 38 with a call to the run-time library routine **memset** to initialize the array:

```
memset(flags, TRUE, sizeof(flags)) ;
```

On the surface everything looks fine, since the run-time function is supported by all compilers and the code is not dependent on the size of the array. Yet when vendor A runs the benchmarks, the tables are again turned, and we are not surprised when we see which compiler comes out on top.

So there you have it—one benchmark with trivial modifications resulted in three different winners. For the record, compiler A was Turbo C 2.0, compiler B was Microsoft C 5.1, and compiler C was WATCOM C 7.0.

A number of subtle lessons are to be learned from this example. Comparing the best execution times of the three compilers shows that the compilers are virtually indistinguishable, differing by no more than a few percent. This is especially significant since the total amount of code in this benchmark amounts to less than 200 bytes. What seems important is that the programmer understand the capabilities of a compiler, exploiting the strong points and working around any limitations. The results say quite clearly that these compilers can perform equally well given equal opportunities to prove themselves.

If there is a lesson to be learned, it is to beware of synthetic benchmarks, especially when the benchmark suite is trivial. Since this popular benchmark could reasonably support three different and conflicting sets of claims, it is difficult to attach much significance to the final results, especially when the test conditions are not completely stated. The solution to the problem is a detailed look at a more sophisticated benchmark, drawn from an actual application.

SORTING OUT CLAIMS

Personal experience has shown that a compiler vendor's claims to highly optimized code generation does not mean that a particular application will run faster than when compiled with a "less optimizing" compiler. Too many other variables influence performance, including run-time libraries and operating system overhead that can overshadow the otherwise excellent code-generation capabilities of a compiler. Not that there aren't applications that are compute bound (making no run-time library calls) and that favor one compiler over another—just that such applications are the exception rather than the rule.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

If this were not the case, the conventional wisdom would have the more optimizing compilers from Microsoft and WATCOM consistently outperforming a compiler like Turbo C, which supposedly lacks some of the more sophisticated optimizations. Yet building an application with each of these compilers and benchmarking a real application will once again yield some surprising results.

ANOTHER TEST

For example, Table 4 shows execution times for Paradigm LOCATE, a medium-size application used to place the output from PC compilers into ROM. Being written in standard C, Paradigm LOCATE has been compiled with Microsoft C, Turbo C, and WATCOM C, providing a more substantial benchmark for measuring compiler performance.

The benchmark runs Paradigm LOCATE twice, once over a modest 40K embedded application and once over a 575K application. Two very-different-size applications are used to determine any effect of file size on the execution time. Table 4 shows that again the execution times are very close, with perhaps a slight edge to the Turbo C and WATCOM C implementations.

While it is certainly possible to make a fuss over the few tenths of a second separating the compilers, a 10-percent difference in performance is not usually noticeable when a human is in the control loop. The more interesting conclusion is to declare the results a tossup and look for other criteria to tip the scales one way or another. It makes much more sense to choose a compiler on the basis of robustness, run-time libraries, debugger, documentation, technical support, or any of a dozen other reasons.

A special observation of the execution analysis of Paradigm LOCATE is that it is an I/O–bound application. Using INSIDE! to partition the execution time shows that more than 50 percent of the execution time was spent in MS-DOS—outside the ability of the compiler-code generator to affect execution time. Under these circumstances, the ability of an optimizing compiler to influence the overall results is reduced, while the importance of bundling efficient run-time libraries with the compiler is increased.

TABLE 1

	Compiler A	Compiler B	Compiler C
small model	161.322	127.618	127.114
large model	161.330	166.818	139.217
total	322.652	294.436	266.331

Sieve test cases (times are in milliseconds)

TABLE 2

	Compiler A	Compiler B	Compiler C
small model	161.324	127.622	127.116
large model	161.326	127.622	139.215
total	322.650	255.244	266.331

Sieve test cases (times are in milliseconds)

TABLE 3

	Compiler A	Compiler B	Compiler C
small model	128.776	129.349	127.551
large model	129.100	168.555	139.227
total	257.876	297.904	266.778

Sieve test cases (times are in milliseconds)

TABLE 4

	Turbo C	Microsoft C	WATCOM C
40K application	3.071	3.240	2.900
575K application	24.112	26.694	24.960

Paradigm LOCATE test cases (times are in seconds)

TABLE 5

Function	Turbo C	Microsoft C	WATCOM C
qsort()	76.419	504.263	380.699
malloc()	33.797	11.755	13.995
free()	32.859	4.254	18.275
write()	2868.981	3023.693	2689.182
memcpy()	9.935	9.755	10.034
strlen()	41.851	25.483	38.375

Run-time library performance (times are in milliseconds)

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Unfortunately INSIDE! cannot easily separate the time spent in the run-time libraries from the time spent in the application. But an examination of functions that we know to be compute bound shows that the code emitted by Microsoft C and WATCOM C was generally faster than the identical code generated by Turbo C, although there are many exceptions where the reverse holds true. Since there is no reason to assume this is not the case for the remaining functions, one can conclude that the Turbo C run-time library functions are more efficient than their Microsoft and WATCOM counterparts.

Where INSIDE! can help is in a look at the source-line timing statistics for calls to run-time library routines. Table 5 contains the execution times for some commonly used C run-time library functions, broken down by compiler. These times were taken from the analysis of a single source module in the source-line-timing mode.

Knowing how the different compilers operate makes it easy to draw conclusions for the above results. Microsoft shows the best execution times for the dynamic-memory-management routines **malloc** and **free**, because it does not go directly to MS-DOS to satisfy each request (as Turbo C does)—it uses a local-memory pool instead. One would also reasonably expect the in-line string operations to favor Microsoft and WATCOM, especially when the strings are short and the overhead of making a function call is a significant percentage of the execution time.

The real mystery is why Borland's implementation of **qsort** is so good. It needed only 6,450 swaps to sort the 531 items in the array, while WATCOM C required 43,273 and Microsoft needed 47,252 swap operations. By shaving nearly half a second from the execution time for the application, Turbo C can afford to be marginally slower in other areas yet still come out on top with the best overall execution time.

ON YOUR OWN

One important consideration not to be underestimated is your own programming skills. While lacking certain statistically significant results, my gut feeling based on my own understanding of popular PC compilers and my own coding skills suggests that optimizing compilers help out programmers writing poor code a great deal more than they help the technically astute programmer writing tight, well-designed code.

This is in part owing to the one optimization missed by all compilers: the optimal selection of an algorithm. A programmer choosing the bubble-sort over the quick-sort algorithm will find that all the code optimizations in the world will not improve the performance of a bubble sort to that of the quick sort when the sort data is randomly organized. Since there is no substitute for a thorough understanding of algorithm design, careful design from the start will contribute much more to the performance of an application than will the code-generation capabilities of the compiler.

All of these compilers are good, and each has its unique advantages that can be used as alternative evaluation criteria. Microsoft C has long been the yardstick by which other compilers compare themselves, and, although becoming dated, it is still a force to be reckoned with. For code size, WATCOM C consistently generates the smallest executable files, albeit at the expense of the longest compile times. For fast compile/link cycles and serious debugging, the Turbo C/Turbo Debugger package just can't be beat, and, as the results showed, Turbo C is an equal match for the other two compilers.

Given the financial wherewithal, a programmer should have a minimum of two of these compilers in his or her toolbox at all times. With the PC C-compiler market being as tremendously competitive as it is, the ultimate winner is, happily, the user. After all, what other programming platform do you know of that supports such a wide diversity of competitive programming tools, all of which can be used to meet the requirements of nearly any application?

LISTING 1

Sieve of Eratosthenes

```
#include  <stdio.h>

unsigned  sieve(void) ;

#define   TRUE      1
#define   FALSE     0
#define   NITER     10
#define   SIZE      8191

char   flags[SIZE+1] ;

void   main()
{
   int   iter ;
   unsigned   primes ;

   printf ("%d iterations: ", NITER) ;
   for (iter = 1; iter <= NITER; iter++)
      primes = sieve() ;

   /* Display the last number of primes found */
   printf("%d primes\n", primes) ;

   /* Exit */
   exit(0) ;
}

unsigned   sieve()
{
   register int   i, k ;
   unsigned   count ;

   /* Start out with zero primes */
   count = 0 ;

   /* Set all flags to TRUE before starting */
   for (i = 0; i <= SIZE; i++)
      flags[i] = TRUE ;

   /* Run the Sieve */
   for (i = 2; i <= SIZE; i++)   {
      if (flags[i])   {
         /* Cancel out all multiples of this prime */
         for (k = i + i; k <= SIZE; k += i)
            flags[k] = FALSE ;

         /* Increment the tally of prime numbers */
         count++ ;
      }
   }

   /* Sieve completed */
   return   count ;
}
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C Programmers

Explore a world of DOS memory

Beyond 640K

Use PowerSTOR to exploit:

- LIM 4.0 or LIM 3.2 Expanded Memory
- AT Style Extended Memory
- Hard Disk Space
- Any Combination

- PowerSTOR is the easy way to use system memory resources. It self-configures at run time, under control of your software.
- PowerSTOR makes system memory resources available to your software as STORheaps. STORheaps can be allocated and de-allocated dynamically. Up to a gigabyte of PowerSTOR memory can be used as one huge STORheap or many smaller STORheaps.
- PowerSTOR uses only 15K in the lower 640K.
- PowerSTOR with K&R standard source code – only $92.50.

Order Toll Free in the US and Canada

1-800-621-5663

Acme Software

P. O. Box 40734, Portland, Oregon 97240 — Phone (503) 235-2816

Number **305** on the Reader Service Card

Confusing Code?

```
#include <stdio.h>
text_count(){int c,nlines,nwords,nchars,inword;
inword=NO;nlines=nwords=nchars=0;while((c
=getchar())!=EOF){++nchars;if (c=='\n')
++nlines;if ((c==' ')||(c=='\n'))inword=NO;else
if(inword==NO){inword=YES;++nwords;}}printf(
"%d %d %d\n",nlines,nwords,nchars);}
```

C It Your Way!

```
#include <stdio.h>

text_count ()
{
     int c, nlines, nwords, nchars, inword;

     inword = NO;
     nlines = nwords = nchars = 0;
     while ((c = getchar()) != EOF) {
          ++nchars;
          if (c == '\n')
               ++nlines;
          if ((c == ' ') || (c == '\n'))
               inword = NO;
          else if (inword == NO) {
               inword = YES;
               ++nwords;
          }
     }
     printf ("%d %d %d\n", nlines, nwords, nchars);
}
```

NOW with **C-Clearly™**, format C source code exactly the way you want it. C-Clearly's context sensitive analysis will format any C program in your own personal or corporate style.

EASY to use, C-Clearly's style templates are a snap to modify, since they resemble C source code you edit into your preferred format. Templates are included for several common styles as well as standard K & R.

LISTINGS can also be created with function names and comments highlighted for improved readability. Listing options include line numbers, headers and/or footers and flow lines.

IDEAL for making obtuse code clear. Allows all of your source code to be presented in a consistent format of your choosing. Also great for code walkthroughs and final documentation listings.

WORKS with all IBM PC, XT, AT, PS/2 and compatibles, with 512K RAM. Automatically processes all include files and pre-processor statements. ANSI-C compatible. Not copy protected.

C-Clearly $129.95

Shipping & Handling USA $5; Canada/Mexico $10; Other Countries $15; CA Residents add sales tax. Visa/MasterCard/COD accepted.

For orders or information call:

1-800-662-8266

V COMMUNICATIONS, INC.

4320 Stevens Creek Blvd., Ste 275, Dept. PJ6

San Jose, CA 95129 (408) 296-4224

Number **341** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

By John Otken

Advanced segments

Advanced techniques require full segment directives

Assemblers and compilers based on the Intel 80x86 object-module format use the concept of "segments" to collect and organize instructions and data. Most programmers find segments complex and confusing. Understanding them is the computer equivalent of a "rite of passage."

The first step to demystifying segments is to come up with a good analogy—office paperwork and file cabinets come to mind. A segment is like a file folder. It has a name similar to the name on the tab of a folder and holds data from different modules, as a folder holds sheets of paper from many sources. Both segments and folders have limited capacity. Segments are organized into classes such as code and data; folders are organized into drawers by customer or invoice. Try visualizing a program as sheets of paper and folders in a file cabinet.

In previous columns, segments were controlled with simplified directives, such as .CODE and .DATA. In this column, we examine segments in more detail and see how to manipulate them with full segment specifications.

Assemblers have four directives for full segment control: SEGMENT, ENDS, GROUP, and ASSUME. The SEGMENT directive opens a segment definition. The first occurrence of this directive requires a few parameters for alignment, combine type, and class name; then, the name and segment keyword are sufficient (see second example below). The ENDS directive closes a segment definition. Segment directives can nest, but only the innermost segment is active.

Compare a short program using simplified segment directives:

```
        .model small
        .stack
        .data
hello   db      'hello$'
        .code
main:   mov     ax,4C00h
        int     21h
        end     main
```

with an equivalent program using full segment directives:

```
_TEXT   segment word public 'CODE'  ; .model small
_TEXT   ends
_DATA   segment word public 'DATA'
_DATA   ends
STACK   segment para stack 'STACK'      ; .stack
        db      1024 dup(?)
STACK   ends
DGROUP  group   _DATA,STACK
        assume  cs:_TEXT,ds:DGROUP,ss:DGROUP

_DATA   segment                         ; .data
hello   db      'hello$'
_DATA   ends
_TEXT   segment                         ; .code
main:   mov     ax,4C00h
        int     21h
_TEXT   ends
        end     main
```

No one-to-one correspondence exists between simple and full segment directives, because the simplified directives are somewhat intelligent about segment definitions and grouping. If the first program had used the .DATA? directive, the equivalent second program would include another segment definition and a longer group declaration:

```
_BSS    segment word public 'BSS'
_BSS    ends
DGROUP  group _DATA,_BSS,STACK
```

Not only are simplified segments shorter than full segments, but the code is much easier to read. Inquiring readers might wonder what value full segments might have. One answer is that complete knowledge of segments is necessary for better programming. Another answer is that some very advanced techniques require full segment directives. These techniques are introduced later in this column.

The full segment version of the sample does not have to be so long. It is not necessary to declare the segments at the head of the program; doing so establishes link order and avoids limitations of one- and two-pass assemblers, such as TASM and MASM.

ALIGNMENT PARAMETER

The first argument to the segment directive specifies the alignment. Possible values are *byte*, *word*, *dword*, *para* (short for paragraph), and *page*. Paragraph alignment on a 16-byte boundary is the default. Simplified segments default to word alignment, except for the stack segment, which defaults to paragraph alignment.

Alignment is effective on that piece of segment generated by the module only. If a segment has pieces from four modules with a mixture of byte and word alignment, each piece aligns according to its own alignment parameter. Gaps, or dead space, are possible between segment pieces.

Dword alignment is not compatible with Intel OMF86. Phar Lap has a nonstandard, 4K alignment in their OBJ formats. Be careful if you mix and match tools from different vendors.

Two uses for alignment are to increase efficiency and to force segments to a zero base. Sixteen- and thirty-two-bit CPUs operate more efficiently when data aligns properly; when data are not

John Otken is the president of Soft Advances, where he develops software for embedded systems when he isn't snowskiing. When the powder is deep, good luck reaching him at 10811 Washington Blvd., Suite 205, Culver City, CA 90232, 213-559-7015.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

$199 Introductory Offer Ends June 30
(Retail Price: $395)

Serious Assembly.

Nobody really takes assembly language seriously. Nobody, that is, except Lotus, WordPerfect, Novell, and everyone else who needs the tightest, fastest code possible.

But assembly is tough . . . unless you have plenty of ready-to-use assembly language routines at your fingertips. That's why industry leaders spend a lot of time and money creating their own, proprietary assembly language libraries.

You could spend your valuable development time building a library; or you can use **Spontaneous Assembly**, the complete **assembly language library** specifically designed for serious assembly language development.

Over 600 functions and macros.
- String manipulation • Memory manipulation
- Near/far/relative heap management
- Doubleword/quadword integer math
- Array management, sorting, and searching
- Character/numeric/string conversion • Buffer management • Date and time manipulation
- File and directory management • File I/O
- Keyboard I/O • Program/environment control
- Critical error management • Screen I/O with windowing – and more. All carefully documented, tested, and ready to use.

Tight, fast code. Fast.
Every routine is hand-coded and hand-optimized (in assembly, of course). And a consistent, register-oriented parameter-passing convention makes these routines remarkably easy to use. With Spontaneous Assembly, coding in assembly is as fast as coding in a high-level language—without the overhead.

Powerful memory model support.
Spontaneous Assembly unleashes the full power of memory models in assembly language. It supports all Microsoft/Borland standard memory models as well as custom models and mixed-model programming. And it gives you complete control over segment/group names and attributes.

Extensive documentation.
A complete reference manual describes every function, macro, and variable in detail. Step-by-step instructions and technical notes explain memory models, library customization, integration with C, and much more.

Spontaneous Assembly.
Serious assembly language programming doesn't need to be hard any more. Now it can be Spontaneous.

See for yourself. Call now to take advantage of our introductory price. Then use Spontaneous Assembly for 60 days. If you're not 100% satisfied, send it back for a full refund!

Introductory Price.
$199 *Plus Shipping & Handling.*
Full source code included. No royalties.
60 day money-back guarantee.
Visa • MasterCard • Discover

Call now!
1-800-ASSEMBLY
Orders • Information • Support

basetwo DEVELOPMENT

For 80x86 based systems running DOS 2.0 or later. MASM 5.1 or TASM 1.0 recommended.
Base Two Development • A Division of Acclaim Technologies, Inc. • 11 East 200 North • Orem, Utah 84057 • (801) 222-9500
Spontaneous Assembly and Base Two Development are trademarks of Acclaim Technologies, Inc.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Their C start-up code depends on the bug; . . . they can't fix it without breaking their compiler.

aligned, the CPU must do an extra memory read or write. Jump, call, and loop instructions are also more efficient when their target address is aligned.

The technique for accessing global variables via BP requires alignment of DGROUP to a 16-byte boundary. Because simplified segments default to word alignment, I define a null segment of class BEGDATA that forces paragraph alignment for DGROUP (see *PJ* Vol, 8.1).

Page alignment (on a 256-byte boundary) does not apply to MS-DOS since DOS loads programs on a paragraph boundary. This argument might be useful in an embedded environment.

COMBINE-TYPE PARAMETER

The next segment parameter is the combine type. It tells the linker how to combine segments with the same name and class. PUBLIC is the most common combine type. The linker merges public segments with the same name and class into one big segment (64K or smaller).

The other combine type common in assembly language programs is STACK. The link "collects" segments of type .STACK together and points the stack field in the EXE header to the end of this segment. There is usually only a single stack segment.

A bug in the Microsoft linker ignores stack-segment grouping. When a program starts, its SS:SP points to the correct place in memory but uses the segment frame and offset instead of the group frame and offset. In typical Microsoft fashion, their C start-up code depends on the bug; so, now they can't fix it without breaking their compiler.

DOS linkers treat segments with combine-type MEMORY like PUBLIC segments; there's no reason to use this type.

When the linker collects segments of combine type COMMON, it superimposes them at the same location. Microsoft detects the presence of floating-point routines with this function. Consider two modules:

```
aseg  segment common 'data' ; module a
word1 dw 1234h
aseg  ends

bseg  segment common 'data' ; module b
word2 dw 5678h
bseg  ends
```

The linker assigns the same memory location to *word1* and *word2*. Link these modules in order "a + b," and *word1* and *word2* point to 5678h. Link them in order "b + a," and they point to 1234h.

The ability to detect when a module links with a program is valuable. The second part of this column introduces a more flexible method for link-time configuration.

To locate a segment to an absolute position in memory, use the AT combine type. The sample code fragment below clears a monochrome display:

```
video segment at 0B000h
video_ram dw 80*25 dup(?)
video ends

        .code
cls:    mov     ax,video
        mov     es,ax
        assume  es:video
        lea     di,video_ram
        mov     ax,0720h
        mov     cx,length video_ram
        rep     stosw
```

DOS linkers ignore data in SEGMENT ATs. You cannot use them to initialize absolute-memory locations.

PRIVATE SEGMENTS

If no combine type appears, the segment is private. The linker does not collect private segments, which are of limited use to programs under MS-DOS. If you want to restrict access to a segment, declare none of its symbols public; unless your program is broken, other modules cannot touch the segment.

CLASS PARAMETER

The final segment parameter is CLASS, which the linker uses to group segments. The three major classes, CODE, DATA, and STACK, are self-explanatory; the other common classes are uninitialized data (BSS) and constant data (CONST).

```
_DATA segment word public 'DATA'      _DATA ends
```

The class name is enclosed by single quotes. When the class parameter is missing, the assembler defaults to a blank or null class name.

Other than in the segment directive, the assembly language program does not use the class name. You cannot define a symbol that conflicts with a class name. Unlike segments, class length is not limited to 64K.

GROUP DIRECTIVE

The GROUP directive permits a programmer to organize data into several segments yet make the segments appear as one to the program. Efficient programming requires the separation of uninitialized and initialized data into individual segments. Consequently, we don't want to load different segments to access these segments. Given the modest stack requirements of most assembly language programs, SS is the most convenient segment register for accessing data. Grouping the data segments and the stack into a single "virtual" segment offer the best of both worlds: organization and single-segment access.

Grouping is a valuable technique to control the position of data in what appears to the program as a single segment. Later in this column, I present two clever uses for grouped segments.

ASSUME DIRECTIVE

If a source file contains full segments and any lines of code, it will require an ASSUME directive. This indicates to the assembler the segment or group in each segment register: CS, DS, ES, and SS. You also can tell the assembler to assume "nothing" about a register. The assume directive implied by ".model small" is:

```
assume cs:_TEXT,ds:DGROUP,ss:DGROUP,es:nothing
```

When the assembler compiles an instruction with a symbolic memory reference, it tries to match the symbol's segment with the contents of a segment

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

register. The assembler checks the default segment register first. If it matches, a segment-override prefix isn't required. If it doesn't match, the assembler checks the other segment registers in the following order: DS, SS, ES, and CS. If it matches the symbol's segment with the assumed contents of a segment register, it generates an appropriate override prefix.

```
assume ds:nothing,ss:DGROUP
lds  si,argv[bp] ; no override prefix
mov  cx,argc     ; auto SS: override generated
```

The default segment for the above LDS instruction is SS. SS contains DGROUP, and *argv* is in DGROUP; a segment-override prefix isn't necessary.

The default segment for the MOV instruction is DS. DS contains "nothing" according to the assume directive. *argc* is in DGROUP. Because DGROUP and "nothing" do not match, the assembler checks SS, which contains DGROUP—a match. The assembler generates an automatic SS:-override prefix.

CONFUSING GROUPS

MASM is notorious for bugs in group handling. If you write much code with MASM, I guarantee many hours of wasted time. In the following example, MASM confuses the segment/group association for the symbol "two." As a result, the address in the "es:two" instruction is different from the address in "ds:two" and "offset two."

```
        dosseg
        .model small
        .data
one     db      1, 15 dup(?)
        .data?
two     db      ?
        .code
main:   mov     ax,DGROUP
        mov     ds,ax
        mov     es,ax
        mov     es:two,2        ; WRONG!
        mov     ds:two,2        ; OK
        mov     bx,offset two   ; OK
        mov     ax,4C00h
        int     21h
        end     main
```

The key to this bug is the assume that ".model small" implies: DS contains DGROUP and ES contains "nothing". It should not effect the segment association for a symbol reference, but it does. Consider a second example:

```
        dosseg
_DATA   segment para public 'DATA'
one     db      1, 15 dup(?)
_DATA   ends
_BSS    segment word public 'BSS'
two     db      ?
_BSS    ends
DGROUP  group   _DATA,_BSS
_TEXT   segment word public 'CODE'
        assume  cs:_TEXT,ds:DGROUP,ss:DGROUP
main:   mov     ax,DGROUP
        mov     ds,ax
        mov     es,ax
        mov     es:two,2        ; WRONG!
        mov     ds:two,2        ; OK
        mov     bx,offset two   ; WRONG!
        mov     ax,4C00h
        int     21h
_TEXT   ends
        end     main
```

In the preceding example, MASM associates DGROUP with the expression "offset two." In this example, MASM associates segment _BSS for the same expression. The only difference in the two examples is the use of full segment directives instead of simplified ones.

You can hack a program into operation with DGROUP: overrides, as Microsoft would have you do. Such a convention obscures algorithms, reduces readability, and makes your code ugly. There is no guarantee you won't miss a line or two and spend hours debugging.

```
mov     ah,9
mov     ds,dgroup_segment[bp]
mov     dx,offset out_of_memory
int     21h
```

A bad scenario might involve a very simple error routine that displays an "out of memory" message. As it happens, you never run out of memory while testing. Send the program to a big customer who loads a huge data file. The program detects the error and jumps to the error routine. Instead of an informative message, the customer gets trash all over his screen.

The problem is worse when using full segment specifications. OPTASM and TASM dutifully emulate MASM's misbehavior, although you can configure OPTASM to work correctly. If you are a MASM user, avoid too much programming with full segments.

MANAGING MASM

With simplified segments, MASM's problems are manageable. A problem occurs only if you use an ES override to access a DGROUP variable. Using the "assume ds:nothing" from the preceding column extends the problem to DS overrides. Presume that DS and ES contains DGROUP, and consider the following:

```
mov  ax,argc          ; OK
mov  ax,ds:argc ; broken if ASSUME DS:NOTHING
mov  ax,es:argc       ; broken
mov  ax,argc[bp]      ; OK
mov  ax,DGROUP:argc   ; OK
```

The only problem instructions are those with forced overrides to segment registers assumed to "nothing". MASM is the only assembler with this problem; OPTASM and TASM work fine.

I would like to thank Randy Sloyer

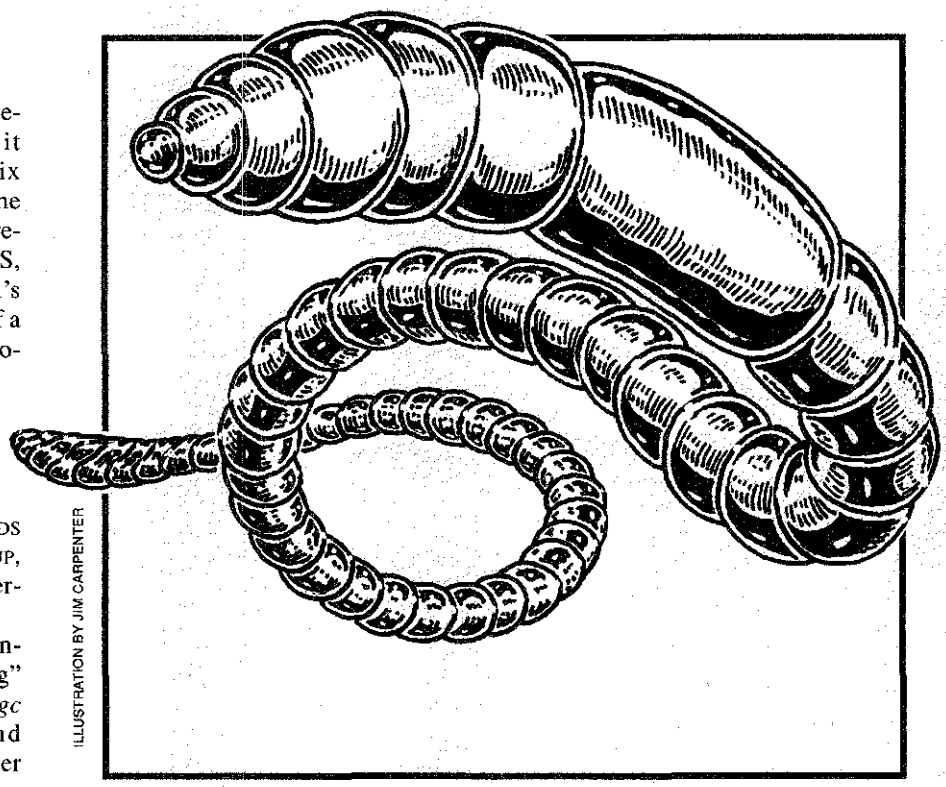


ILLUSTRATION BY JIM CARPENTER

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

for pointing some of this out to me. We OPTASM devotees tend to forget the worries of MASM and TASM users.

I advocate a coding style that avoids assume directives. In my programs, symbols refer to either code or data. If they refer to code, they are in _TEXT and the assembler assumes CS contains _TEXT. If they refer to data, they are in a DGROUP segment and the assembler assumes SS contains DGROUP.

DS and ES are constantly changing between DGROUP and dynamic storage. To be rigorous, I would have to litter my code with **assume** directives. Since I do not want the assembler to generate automatic segment-override prefixes, I have little use for **assumes**.

The Software Engineering Store

C

Borland Turbo C v2 Prof. $175
Blaise C Tools Plus/6.0 ..... $119
Clear Clear Plus for C $169
Creative Vitamin C $169
Genus PCX Prgmr Toolkit $155
manipulate ".pcx" graphics files
Gimpel PC Lint $99
Greenleaf Comm Library $179
Island Systems C Menu $129
Microsoft C v5.1 ..... $325
Microsoft Quick C $73
Novell Btrieve $199
Sequitor Codebase 4 ..... $249
read, write & create dBase III and Clipper files
Wordcraft C Workshop $69
C training manual & software

Other Tools

Digitalk Smalltalk/286 ..... $164
Burton Tlib $89
source code management system
California Scientific Brainmaker $159
neural network development system
Genus PCX Toolkit $155
Helios Proteus $169
Matrix Layout v2 ..... $175
Microsoft Macro Assembler $111
Microsoft Windows SDK v2.1 ..... $349
Mortice Kern MKS Toolkit v3 ..... $219
KORN shell & 130 UNIX™ -like utilities
Periscope I 512K v4 ..... $499
Qualitas 386-Max $69
80386 memory manager
Quarterdeck API Ref & C Library .. $179
Solution Systems Brief ..... $call
Turbo Power Turbo Professional $99
Whitewater ACTOR $449
High productivity development for MS Windows

C++ Version 2

Zortech C++ is the only true C++ compiler available for MSDOS. Now includes AT&T C++ v2 features including multiple inheritance and type safe linking. C++ debugger lets you view objects instead of translated C code.
C++ v2 ..... $175
C++ Debugger $129
C++ Developer's Edition $399

The Software Engineering Store

1-800-443-7176

In Calif: (415)957-0111

Visa • MasterCard • AMEX
COD • Check • P.O.

- Most orders shipped within 24hrs
- Free shipping on orders over $100

657 Mission Street, #300-PJ
San Francisco, CA 94105

Free Newsletter

Call or write for a catalog and your free subscription to our newsletter.

Prices subject to change without notice. ©1989 TSF. All Rights Reserved.

Number 98 on the Reader Service Card

An alternative viewpoint comes from Randy Sloyer, who sent me a code fragment that shows how to use **assumes** and dummy segments to document code. In the example below, INBUF and OUTBUF are dummy-segment definitions; they are not real segments. The code loads dynamic-storage segments into DS and ES and then uses an assume directive to assign a symbolic name to the registers describing what kind of storage they point to. He overrides each instruction with the segment name and lets the assembler figure out the required override prefix:

```
INBUF   segment
INBUF   ends
OUTBUF  segment
OUTBUF  ends

start   proc    near
        mov     es,[seg_inbuf]
        mov     ds,[seg_outbuf]
        assume  ds:INBUF,es:OUTBUF

        mov     ax,INBUF:[in_point]
        mov     OUTBUF:[out_point],ax
        ...
start   endp
```

You may find this style helpful. If you have trouble tracking segments, I recommend experimenting with it.

OBJECT MODULE FORMATS

For a better understanding of segments, you should study the object file format. It will make you a better programmer. The standard reference guide is Intel's *8086 Relocatable Object Module Formats*. The order number is 121748-001; Intel's phone number is 800-548-4725. This book cures insomnia.

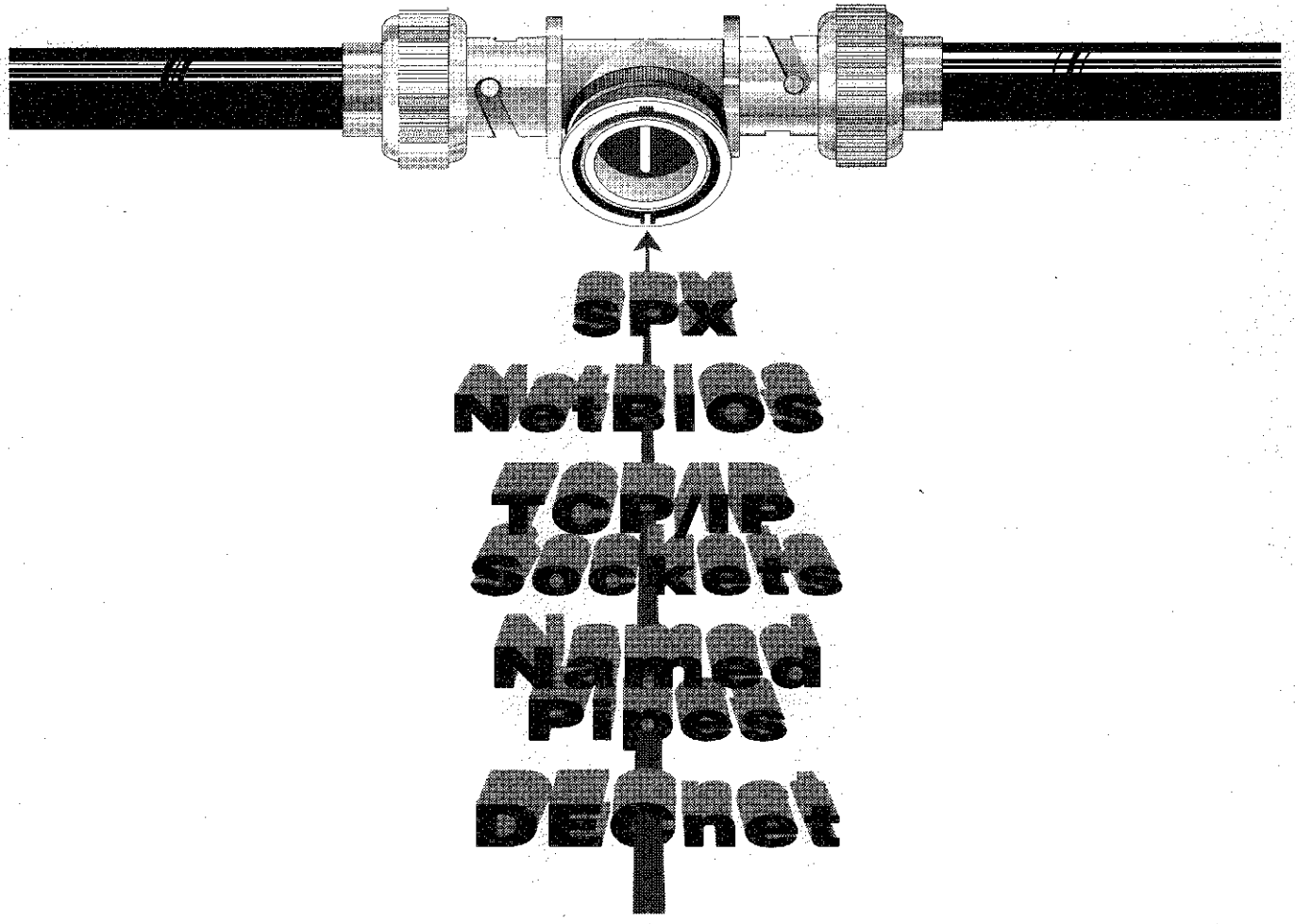
Microsoft OBJ is a subset of Intel OMF86, with a few exceptions. Those exceptions include 0xB2 records for communal variables, DWORD segment alignment in SEGDEF records, and 386 support. Microsoft does not use Intel debug records.

There are several OBJ-dumping programs available. Borland includes a very good program called TDUMP with Turbo Debugger. TDUMP dumps OBJ files, Codeview EXE, and Turbo Debugger EXE files. I use it all the time. Learning something about the OBJ documents is important if you want to understand the DOS linker.

In spite of its massive size, the DOS linker is really not a complicated program. The basic process of linking

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Are networks creating obstacles for your applications?



Writing network communications software for your applications is difficult enough. Writing it for multiple networks or to connect applications running on different operating systems and hardware platforms is just too much. Too much time. Too much money. And too much energy searching for the networking software expertise you need to overcome your multi-vendor computing obstacles.

At Netwise, we believe your time and money are best spent at the applications level, building solutions for today's network computing environments — not struggling with writing low-level networking code.

The Netwise® RPC TOOL™ lets you breeze past the obstacles by providing access to a common software platform for the development of distributed applications. You write the application, then specify the network. The RPC TOOL automatically generates distributed processing software for you. All of it. Every time. And does it TODAY — for NetBIOS, NetWare, LAN Manager, DECnet, and TCP/IP networks. No waiting on promises. No vaporware. No excuses.

Banyan, Novell, Sun, 3Com and other major software and hardware suppliers have endorsed this technology as a standard for building distributed applications. Now it's your turn.

Call us for a FREE educational diskette. Or order a PC version of the RPC TOOL for a FREE thirty-day trial. Save your struggling for obstacles worthy of the effort.

Use the Netwise RPC TOOL once, and never write network communications code again.

Netwise is a registered trademark and RPC TOOL is a trademark of Netwise, Inc. Banyan is a registered trademark of Banyan Systems Incorporated. DECnet is a trademark of Digital Equipment Corporation. Novell is a registered trademark and NetWare is a trademark of Novell, Inc. Sun is a registered trademark of Sun Microsystems, Inc. 3Com is a registered trademark of 3Com Corporation.

NETWISE®

Call us. 800/733-7722
2477 55th Street,
Boulder, Colorado 80301
USA

See Us at UniForum, Booth #1347

Call me, I'm Interested: Circle 378 **Send Literature: Circle 278**

REPRODUCED BY UNIFORUM
ELECTRONIC REPRODUCTION PROHIBITED

Commenting Disassembler!

SOURCER™ 486

■ SEE HOW PROGRAMS WORK
■ EASILY MODIFY PROGRAMS

SOURCER™ creates detailed commented source code and listings from memory and executable files directly suitable for reassembly. Built in data analyzer and simulator resolves data across multiple segments and provides detailed comments on interrupts and subfunctions, I/O ports and much more. Includes a definition file facility to include your own remarks and descriptive labels, force data types, and more. Complete support for 8088 through 80486, 8087 to 80387, and V20/V30 instruction sets.

We welcome comparisons with any other product, because no product comes close to the ease of use and output clarity of SOURCER. "Sourcer is the best disassembler we've ever seen." PC Magazine 1/17/89 page 101.

BIOS SOURCE

for PS/2, AT, XT, PC and Clones

■ CHANGE AND ADD FEATURES
■ CLARIFY INTERFACES

The BIOS Pre-Processor™ with SOURCER provides the first means to obtain accurate legal source listings for any BIOS! Identifies entry points with full explanations. Resolves PS/2's multiple jumps for improved clarity. Provides highly descriptive labels such as "video_mode" and much more. Fully automatic.

ASM ProPak™

■ ASSEMBLY TOOLS PACKAGE

Save $50 with the ASM ProPak™ assembly tools package. It includes Sourcer with the BIOS Pre-Processor; Unpacker,™ for unpacking packed EXE files; ASMtool,™ to automatically generate flowcharts and tree diagrams from assembly source code; and View-It,™ to quickly view files of any size and see up to 8 times more information on a single screen using EGA/VGA equipment.

Sourcer–Disassembler	$ 99.95
BIOS Pre-Processor	49.95
Unpacker–Unpacks packed files	39.95
ASMtool–Source code analyzer	89.95
View-It–Zoomable file viewer	69.95
ASM ProPak–All Above (save $50)	299.75
Sourcer w/BIOS–(save $10)	139.95

Shipping & Handling: USA $5; Canada/Mexico $10; Other Countries $15; CA Residents add sales tax. Visa/MasterCard/COD accepted.

NOT COPY PROTECTED
30-DAY MONEY-BACK GUARANTEE

If within 30 days of purchase you find our product does not perform in accordance with our claims, call our customer service department and we will gladly arrange a refund.

For orders or information, call:

1-800-662-8266

V COMMUNICATIONS, INC.
4320 Stevens Creek Blvd., Suite 275, Dept. PJ
San Jose, CA 95129 (408) 296-4224

Number **125** on the Reader Service Card

involves two passes. The first pass scans the object files for segment definitions. After this pass, the linker knows the size and position of all segments. A second pass fills the segments with data and fixes the symbolic references between segments.

DOS SEGMENT ORDER

Once the linker knows the sizes of the various segments, it must order them. In the absence of the DOSSEG switch, the linker strategy is first come first serve, with the exception that the linker permits segments with the same class name to cut in line. A reasonable way to ensure segment order is to declare all segments in the first module in the order they are to appear.

When DOS segment order (dosseg) is enabled, the linker orders segments into three major sections. The first section includes all segments with a class name that ends in CODE. One expects the contents of these segments to be instructions. Within the code section, segments are ordered by class name in the sequence in which they appear in the OBJ files.

The second section includes all other segments not in DGROUP. As before, segments are ordered by class name in the sequence in which they appear in the OBJ files. This section is empty for small-model assembly language programs.

The last section includes all segments in DGROUP. The first of four subsections is all segments of class BEGDATA. You should recall our use of an empty segment named NULL of class BEGDATA that forces paragraph alignment of DGROUP. Microsoft C uses BEGDATA to detect null-pointer assignment. Turbo C does not appear to use BEGDATA.

The second subsection includes all remaining segments with class names other than BEGDATA, BSS, and STACK—specifically, segments of class DATA and CONST.

The third subsection contains segments with class BSS. Uninitialized variables go here. Most start-up procedures zero this class when a program starts. The linker automatically defines symbols *_edata* at the start of this subsection and *_end* at the end.

The last subsection includes segments of class STACK. Positioning the BSS and STACK segments at the end of a program saves space in the EXE file.

LINKING WITH FULL SEGMENTS

Though full-segment directives are verbose and problematic, they offer capabilities unavailable to high-level-language programs. Building data structures and procedures with the linker is such a capability.

My first and favorite example of this technique is table building. Large programs often require debugging commands and routines for testing, but debugging code does not belong in the customer release. With "link tables," you can add a module to the link-response file, test the program, and build the customer release without reassembly or changes to any lines of code. Or you may want to build different releases of a product for sale to different customers.

To build a link table, you define a new segment for table contents. When linker runs, it combines all the segments together in the program. Another segment delimits the table. You define both segments in the first link module.

See Listing 1 for a simple sample program. This program echoes command line arguments that appear in a link table. The program defines segments CMD_DAT and CMD_END. ASCIIZ-table entries go in CMD_DAT. A 0 byte in segment CMD_END delimits the table. Both segments are declared as part of DGROUP. When this program links, any modules can add entries to the table. See Listing 2 for an example; notice that the segments are byte aligned.

Assemble and link Listing 1, and execute the program with command line arguments *one two three four five*. The program displays "one two".

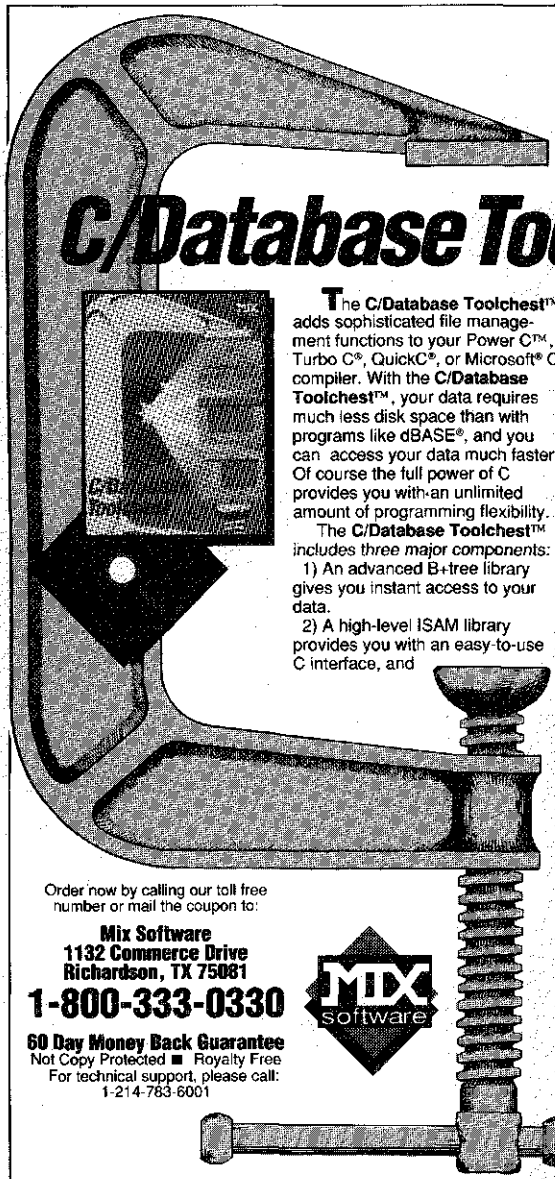
```
C>asm test;
C>link test,,,lib;
C>test one two three four five
one two
```

Assemble Listing 2 and relink. Enter the same command line. This time, the program displays "one two three four".

```
C>asm test2;
C>link test+test2,,,lib;
C>test one two three four five
one two three four
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Turn your favorite C compiler into a powerful database manager with the

C/Database Toolchest

The **C/Database Toolchest™** adds sophisticated file management functions to your Power C™, Turbo C®, QuickC®, or Microsoft® C compiler. With the **C/Database Toolchest™**, your data requires much less disk space than with programs like dBASE®, and you can access your data much faster. Of course the full power of C provides you with an unlimited amount of programming flexibility.

The **C/Database Toolchest™** *includes three major components:*

1) An advanced B+tree library gives you instant access to your data.

2) A high-level ISAM library provides you with an easy-to-use C interface, and

3) A complete database manager (with C source code included) shows you how to create impressive applications.

You also receive a comprehensive 350 page manual and a utility for converting dBASE® files.

The **C/Database Toolchest™** supports features that you'd expect to find only in products costing ten times as much. Advanced features include variable length records, variable length keys, multiple keys per index, and multiple indexes stored in a single file. Your data files can contain an unlimited number of records, and each record can be as large as 32K bytes in length.

About the only thing that the **C/Database Toolchest™** doesn't do is cost you a lot of money. We've kept our price low so you can manage your budget as easily as your data.

Now Only $19.95!

Order Coupon

Name_____
Street_____
City_____
State_____ Zip_____
Telephone_____
Paying By____ Money Order____ Check
____ Visa ____ MC ____ AX ____ Disc.
Card#_____
Exp. Date_____
Disk Size____ 5¼" ____ 3½"

Qty.	Product	Price	Subtotal
____	C/Database Toolchest	$19.95	____
____	C/Database Library Source. B+tree & ISAM library source code	$10.00	____
	Add Shipping ($5 USA, $20 Foreign)		____
	Texas Residents Add 8% Sales Tax		____
	Total Amount of Your Order		____

Order now by calling our toll free number or mail the coupon to:

Mix Software
1132 Commerce Drive
Richardson, TX 75081

1-800-333-0330

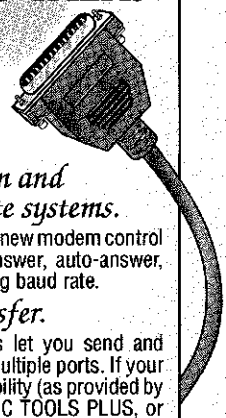
60 Day Money Back Guarantee
Not Copy Protected ■ Royalty Free
For technical support, please call:
1-214-783-6001

C/Database Toolchest and Power C are trademarks of Mix Software. QuickC and Microsoft C are registered trademarks of Microsoft. Turbo C is a registered trademark of Borland. dBASE is a registered trademark of Ashton-Tate.

Number 266 on the Reader Service Card

WE HAVE WAYS TO MAKE YOU TALK.

(And let you listen in, too)



with tools from Blaise Computing.

Talk to your modem and connect with remote systems.

Go online with ease with our new modem control routines. Initialize, dial, answer, auto-answer, and auto-adjust for incoming baud rate.

Let's talk file transfer.

XModem/YModem routines let you send and receive multiple files over multiple ports. If your program has ISR/TSR capability (as provided by our POWER TOOLS PLUS, C TOOLS PLUS, or Turbo C TOOLS), you can talk file transfer in the background while your foreground process talks data input, or database query, or whatever.

The file transfer capabilities include 1K packets, CRC error checking, YModem (multi-file transfers with file name and size preserved), automatic adaptation to incoming packet size and error detection method. And we're talking fast — all file transfer routines have been optimized for maximum throughput.

Let's talk basics.

C ASYNCH MANAGER™ and **ASYNCH PLUS™** retain the features which have made them the libraries of choice for asynch projects — buffered interrupt-driven input and output to multiple COM ports with speeds up to 19200 Baud, XON/XOFF protocol, hardware handshaking and much more.

C ASYNCH MANAGER supports Microsoft C, QuickC and Turbo C. ASYNCH PLUS supports Turbo Pascal and QuickPascal. Both products require DOS 2.00 or later and an IBM PC, XT, AT or PS/2 machine or true compatible.

Let's talk price.

Why pay more? Our ASYNCH packages are just $189, including complete source code, a comprehensive reference manual with extensive examples, sample programs and online help.

Turn that serial cable into a party line with View232!

Debugging serial communications doesn't have to leave you talking to yourself. **View232™** turns your computer into a serial data analyzer that lets you listen in on the conversation between any two serial devices — and this party line provides a transcript of what was said!

View232 displays the data as it flows in both directions. Save a whole transmission in a buffer, then browse through or search the buffer for a pattern or specific character. And you can save the data to disk or print it for later study.

VIEW232 is easy to use. And we supply the cable, all for just $189!

Let's talk track record.

Blaise Computing has produced a collection of tools that are unsurpassed for reliability, flexibility, and ease of use.

And, we're talking guaranteed.

If during the first 30 days you're not completely satisfied, we'll refund your money.

Call (800) 333-8087 today!

BLAISE COMPUTING INC.
2560 Ninth Street, Suite 316
Berkeley, CA 94710
(415) 540-5441
FAX (415) 540-1938

Trademarks are property of their respective holders

Number 2 on the Reader Service Card

CHANGE OF ADDRESS?

Please send us your new address along with your previous ZIP code, using this handy form— Thanks.

NAME_____

COMPANY_____

ADDRESS_____

CITY_____ STATE_____

ZIP CODE_____ OLD ZIP CODE—_____

TELEPHONE_____

Please mail to:
Programmer's Journal
PO Box 70387
Eugene OR
97401-0121

Or call toll free
800-234-0386

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
        title   listing one - link time table
        include asm.inc

        .stack
        .data
        .data?
        extrn   argv:dword,argc:word

CMD_DAT segment byte public 'CONST'
command_table label byte
        db      'one',0,'two',0
CMD_DAT ends
CMD_END segment byte public 'CONST'
        db      0
CMD_END ends
DGROUP  group   CMD_DAT,CMD_END

        .code
        extn    outchr_asciiz,outchr_space,exit_program,startup

;;      main
;
main    proc
        mov     ax,@data                ; compensate for MS-LINK bug
        mov     ss,ax
        mov     sp,offset stack

        mov     ax,sp                   ; init segments, argv, argc, etc
        call    startup

        mov     cx,argc[bp]
        jcxz    mai3                    ;  if no arguments

        mov     ax,@data
        mov     ds,ax

        mov     bx,0
mai1:   lea     si,command_table

        les     di,argv[bp]
        mov     di,es:[di+bx]
        add     bx,2

        call    search_asciiz_table
        jc      mai2
        call    outchr_asciiz
        call    outchr_space

mai2:   loop    mai1

mai3:   jmp     exit_program
main    endp


;;      search asciiz table
;
;       entry   DS:SI   asciiz table
;               ES:DI   search string
;       exit    SI      matching table entry
;               Cf      if search string not found
;       uses    AX
;
search_asciiz_table proc
        jmp     sat2

sat1:   call    strcmp                  ; compare search string with table
        je      sat3                    ;  if match
        call    strskp                  ;  else skip table entry

sat2:   cmp     bptr [si],NULL_CHAR     ; check for end of table
        jne     sat1                    ;  if more table entries
        stc                             ;  else indicate string not found

sat3:   ret
search_asciiz_table endp


;;      strcmp
;
;       entry   DS:SI   string 1
;               ES:DI   string 2
;       exit    Zf      if equal (Cf=0)
;               Cf      if string 2 > string 1
;       uses    AX
;
strcmp  proc
        pushm   di,si
scp1:   lodsb
        mov     ah,es:[di]
        inc     di
        or      ax,ax
        jz      scp2                    ; if end of both strings (equal)
        cmp     al,ah
        je      scp1                    ; if strings equal so far
scp2:   popm    si,di
        ret
strcmp  endp


;;      strskp
;
;       entry   DS:SI   string pointer
;       exit    SI      updated to byte past NULL
;       uses    AX
;
strskp  proc
        lodsb
        cmp     al,0
        jne     strskp
        ret
strskp  endp

        end     main
```

LISTING 2

```
CMD_DAT segment byte public 'CONST'
        db      'three',0,'four',0
CMD_DAT ends
        end
```

LISTING 3

```
        title   link time code
        include asm.inc

        .stack
        .data
pre_terminate_text      db      'Pre-terminate code active',13,10,'$'
        .data?
        extrn   argv:dword,argc:word

terminate_address       dd      ?

        .code

_TERM1  segment byte public 'CODE'
_TERM1  ends
_TERM2  segment byte public 'CODE'
_TERM2  ends
CGROUP  group   _TEXT,_TERM1,_TERM2

        .code
        assume  cs:CGROUP
        extn    exit_program,startup,outchr_registers

;;      main
;
main    proc
        mov     ax,@data                ; compensate for MS-LINK bug
        mov     ss,ax
        mov     sp,offset stack

        mov     ax,sp                   ; initialize segments, argv, argc,
etc
        call    startup

        mov     ah,30h                  ; check dos version
        int     21h
        cmp     al,3
        jb      mai1                    ;  if before DOS3

        mov     ah,62h                  ; get PSP (requires DOS3)
        int     21h
        mov     ds,bx
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

This program requires the include file and libraries developed in earlier columns. You can assemble the program with MASM, TASM, or OPTASM, but you may have trouble using TLINK. It does not seem to support *_edata* and *_end*.

You also can use link tables to organize data within a single module. Say you want to search a table of numbers with the REP SCASW instruction, but the table has several numbers per entry and you want to program the table one entry per line. You can define a macro that opens a segment for the search key, defines the word, closes the segment, and defines the rest of the table entry. Keep the macro definition near the table, not in an include file. See Listing 5 for an example.

LINKING CODE FRAGMENTS

Listings 3 and 4 contain a sample program that links code fragments into a procedure at link time. A real-world use for this technique is initialization code necessary for an optional module (the module may contain debugging commands linked into the command table). Another use is termination-cleanup code.

For example, interrupt-communication routines must disable interrupts from the COM port and restore interrupt vectors before the program terminates. A program can terminate in many ways; for example, the user keys a <Ctrl-Break>, or a floppy drive door is open. If a programmer wants to switch between polled input/output and interrupt routines, the termination-cleanup code can get complicated.

Listings 3 and 4 show how to build procedures with the linker. This example builds a terminate procedure that executes when a program terminates by any means.

"Link procedures" use the same strategy as link tables: Define new segments for the procedure body and end. In the example, segment _TERM1 contains the procedure body. Any module can add code fragments to the procedure by defining instruction within this segment. Segment _TERM2 holds the procedure end that passes control to DOS for final termination.

The program does not do much beside display console messages as the program terminates. Assemble both modules and link the program with and without the second module:

```
C>asm term1;
C>asm term2;
C>link term1,,,lib/map;
C>link term1+term2,term2,,lib/map;
```

Examine the MAP files after linking. The _TERM1 and _TERM2 segments should follow the _TEXT code segment. Run the two programs, and compare the console output.

By the way, exercise caution with the termination procedure—the normal program environment is not available. In particular, the stack pointer no longer points into the program address space; so BP access to static variable is not possible without changing the stack pointer, which is not recommended. The stack size isn't known, and DOS may already have closed any open files (an exercise for the reader).

SOFTWARE DEVELOPMENT '90

There were no big surprises at Miller Freeman's SD '90 show last February. Microsoft is still struggling with a new version of MASM. Version 5.1 is already two years old and ranks a distant third behind OPTASM and TASM. I have no information on when they expect to release version 6.

Borland announced a new version of TASM for release in March. It has "**nop** squashing," a possible euphemism for N-pass assembly à la OPTASM. I will try to review it in my next column here in *PJ*.

I did learn a new reason for programming in assembly language: ecology. The connection is quite simple. Smaller assembly language programs require fewer memory chips. Chip manufacturing uses large quantities of chlorofluorocarbons (CFCs). CFCs eat huge holes in the ozone layer. Smaller programs imply more ozone. High-level languages are clearly a corporate plot to sell more suntan lotion. Save the ozone—outlaw COBOL!

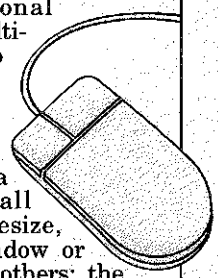
Code for this column is available on many bulletin board systems, including my own. The number is 213-559-1449. I sacrifice no additional ozone molecules for this service—the BBS is a retired IBM PC.

6.0 AND COUNTING!

Integrate sophisticated features into your Microsoft C and QuickC applications with

C TOOLS PLUS/6.0™

C TOOLS PLUS version 6.0 is filled with many advanced routines for developing high-powered C applications, including: virtual, stackable menus and windows with full mouse support and optional "drop shadows"; multiple virtual pop-up help screens; a miniature multi-line editor for gathering user responses in a robust fashion; a single function call which can move, resize, and promote a window or menu on top of all others; the ability to update covered windows automatically when they are written to; support for EGA, VGA, and MCGA text modes including 30-, 43-, and 50-line modes; support for the enhanced (101/102 key) keyboard.



All this and more for only $149!

C TOOLS PLUS/6.0 also contains functions for writing interrupt service routines; creating pop-up memory resident applications; general memory "peeks" and "pokes"; access to the DOS PRINT utility; as well as many other general utility functions and macros.

COMPLETE PROFESSIONAL PACKAGE.

Blaise Computing's function libraries offer easy to use solutions to your programming needs. You get source code, complete sample programs, and a comprehensive reference manual with extensive examples. Supports QuickC and Microsoft C 5.0 and later.

30 DAY GUARANTEE.

If during the first 30 days you are not completely satisfied, we'll refund your money.

Other powerful products from Blaise Computing

C ASYNCH MANAGER™	$149
ASYNCH PLUS™	$149
CHART™	$149
SCREEN SCULPTOR™	$149
Turbo C TOOLS™	$149
POWER TOOLS PLUS™	$149
POWER SCULPTOR™	$149

Call today for more information or to order.

(800) 333-8087

BLAISE COMPUTING INC.

Number 15 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        les     ax,ds:[0Ah]             ; save current terminate address
        mov     wptr terminate_address[bp],ax
        mov     wptr terminate_address[bp+2],es

        mov     ds:[0Ch],cs             ; set new termination address
        mov     wptr ds:[0Ah],offset pre_terminate_code

mail:   jmp     exit_program
main    endp

_TERM1  segment
        assume  ds:@data,es:nothing,ss:nothing
pre_terminate_code:
        mov     ah,9
        mov     dx,@data
        mov     ds,dx
        lea     dx,pre_terminate_text
        int     21h
_TERM1  ends
_TERM2  segment
        mov     ax,@data
        mov     ds,ax
        jmp     terminate_address
_TERM2  ends

        end     main
```

LISTING 4

```
        .model small
        .data
listing4_text   db      'Listing 4 terminate code',13,10,'$'

_TERM1  segment byte public 'CODE'
        mov     ah,9
        mov     dx,@data                ; display terminate message
        mov     ds,dx
        lea     dx,listing4_text
        int     21h
_TERM1  ends

        end
```

LISTING 5

```
        title   listing five
        include asm.inc
        .stack
        .data
one     db      'one$'
two     db      'two$'
three   db      'three$'

table   macro p1,p2
    _TABLE segment
        dw      p1
    _TABLE ends
        dw      p2
        endm

_TABLE segment word public 'DATA'
search_keys     label word
_TABLE ends
DGROUP  group   _TABLE

table_offsets   label word
        table   1, one
        table   2, two
        table   3, three

        .code

;;      main
;
main    proc
        mov     ax,@data
        mov     ds,ax
        mov     es,ax
        mov     ss,ax
        mov     sp,offset stack
        mov     bp,0

        mov     ax,2                    ; search for 2 in search_keys
        mov     cx,3                    ;  (table has 3 entries)
        lea     di,search_keys
        repne   scasw
        jne     mail                    ;  if not found

        sub     di,offset search_keys   ; get corresponding
                                        ;  offset from table
        lea     si,[di-size search_keys]
        mov     dx,table_offsets[bp+si]

        mov     ah,9                    ; display matching string on console
        int     21h

mail:   mov     ax,4C00h
        int     21h
main    endp

        end     main
```


AUTO-HYPHENATION

HY-PHEN-EX is a hyphenation expert system for American English packaged as a C language function. HY-PHEN-EX has over 4800 rules to identify the places where a word may be divided between lines. HY-PHEN-EX comes with full C language source.

HY-PHEN-EX costs only $89.
To order, or for more information, call 415-680-1964 or write to:

GeoMaker Software
P.O. Box 273124
Concord, CA 94527-3124

Number 355 on the Reader Service Card

PJ BBS

On disk from:

Programmer's Journal
P. O. Box 70167
Eugene, OR 97401-0110
Call toll free:
800-234-0386
In Oregon call:
503-747-0800
All Code Disks $10.00 each.
To order, see ad on page 88.
PJ Code DiskScriptions are available for only $39.95.
To order, see ad on page 8.

On BBS from:

Buffer Board
538 Cardinal Way
Springfield, OR 97477
(Voice) 503-741-1414
(Data) 503-747-7636

EDI
1332 Hermosa Ave. #7
Hermosa Beach, CA 90254
213-374-3478
Call for your local access number.

Generation Five Inc.
12061 Tech Road
Silver Spring, MD 20904-9998
(Voice) 301-680-3600
(Data) 301-680-9300
Accessible via PC Pursuit, Washington, D.C.
From Files submenu, select "P" for Personal Downloads.

Gilmore Systems of Southern California
5447 Indian Hills Drive
Simi Valley, CA 93063
(Voice) 805-582-1360
(Data) 805-582-9306

Ithaca Street BBS
1145 Ithaca
Boulder, CO 80303
(Voice) 303-494-8865
(Data) 303-494-8868

Logical Alternatives Incorporated
Calder Square,
P.O. Box 10674
State College, PA 16805-0674
(Voice) 814-234-8088
(Data) 814-234-1881

Mnematics Videotex
P.O. Box 19
Sparkill, NY 10976-0019
(Voice) 800-322-3633
(Voice) 914-365-0184 in N.Y.
Mention *PJ* when you call.

PCS-49 on CompuServe
Section 10
Sponsored by:
MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170
(Voice) 703-450-4181

The Silicon Zone
820 Tivoli Circle #203
Deerfield Beach, FL 33411
(Voice) 305-360-9362
(Data) 305-360-9376

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

People are talking about us.

F77L-EM/32

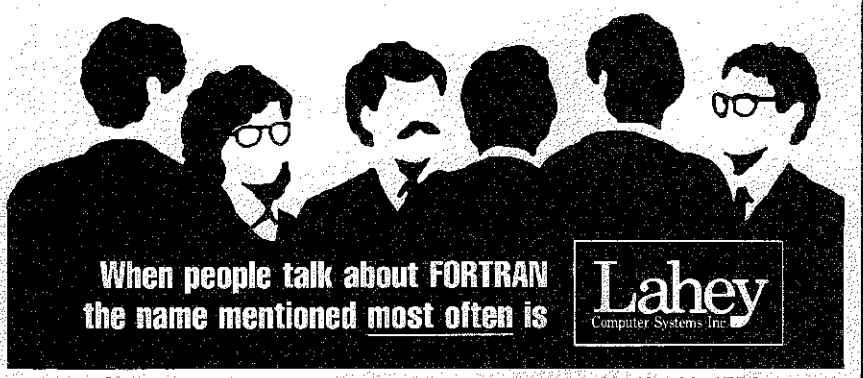
Port 4GB mainframe programs to 80386s with this 32-bit DOS-Extender compiler. The Winner of *PC Magazine's* 1988 Technical Excellence Award just got better. New Version 3.0 and OS include: Editor, Make Utility, Virtual Memory Support, DESQview Support, New Documentation and Free Unlimited Runtime Licenses. F77L-EM/32 $895 OS/386 $395

F77L

The compiler of choice among reviewers and professionals. Includes a Debugger, Editor, Profiler, Linker, Make Utility, Weitek and 386 Real-Mode Support, Graphics. $595

Lahey Personal Fortran 77

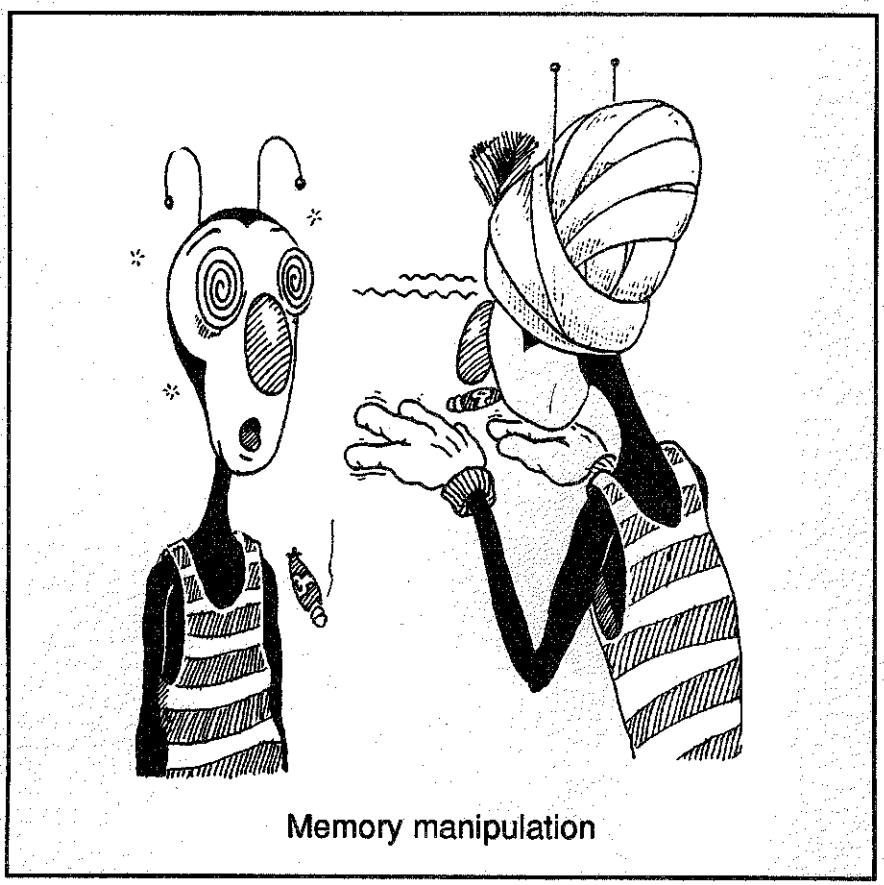
New Version 3.0: Full ANSI 77, Debugger, Editor, Linker, Library Manager, Microsoft and Borland C interfaces, 400 page Manual, Unbeatable Price. $99



Contact us to discuss our products and your needs. **(800) 548-4778**
Lahey Computer Systems, Inc. P.O. Box 6091, Incline Village, NV 89450
Tel: (702) 831-2500 FAX: (702) 831-8123 Tlx: 9102401256

FORTRAN IS OUR FORTE

Number **44** on the Reader Service Card



Memory manipulation

PAINT POWERFUL DATA-ENTRY SCREENS WITH POWER SCREEN™

Design screens and menus with PAINT™, placing input fields and messages precisely where you want them to appear in your final application. PAINT stores screen definitions in a single file — which can be modified without recompiling or relinking your application. Your program manages screen display and I/O through the POWER SCREEN Runtime Library. All essential features are supported, including:

◆ Block multi-field or single-field I/O ◆ Automatic range checking ◆ Configurable editing/menu key definitions ◆ Application context-sensitive, cross-referenced help ◆ Virtual/automatic scrolling screens within viewports ◆ Multiple, overlapping viewport display ◆ Plus much, much more!

The Runtime Library can be linked directly with Turbo Pascal applications, or installed as memory resident.

All this for just $149.00

Includes complete sample programs, a comprehensive reference manual, and the Norton Instant Access program and guides to assist you during program development. And, we offer a 30-day money back guarantee. Supports Microsoft C/QuickC, Turbo C, Turbo Pascal, QuickPascal and QuickBASIC.

Other powerful products from Blaise Computing:

C ASYNCH MANAGER™	**$189**
ASYNCH PLUS™	**$189**
View232™	**$189**
C TOOLS PLUS™	**$149**
Turbo C TOOLS™	**$149**
POWER TOOLS PLUS™	**$149**
POWER SEARCH™	**$149**

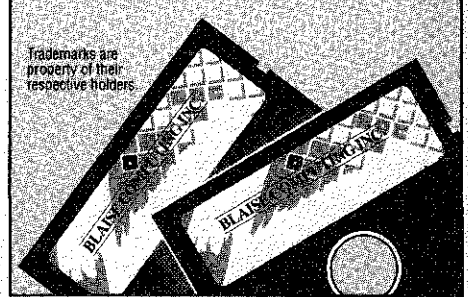
Call today for more information

(800) 333-8087 Fax (415) 540-1938

BLAISE COMPUTING INC.

2560 Ninth Street, Suite 316 Berkeley, CA 94710 (415) 540-5441

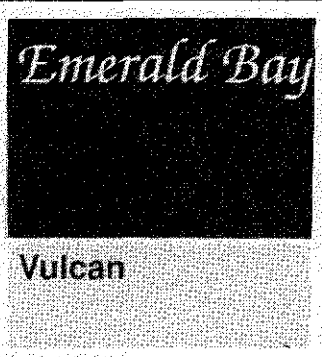
Trademarks are property of their respective holders.



Number **14** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Over 2,500 Software Tools



Emerald Bay Vulcan
by Ratliff Software

Emerald Bay's Vulcan is a next generation dBASE. Both a DBMS language, interpreter and compiler,Vulcan includes user-defined functions, nested reads, mouse-support, save and restore screens, and multi-dimensional arrays. If you know the dBase language, you know Vulcan, whose programs are smaller and easier to maintain. The compiler is in the performance class of Clipper, Foxbase, and Quicksilver.
List: $395 Ours: $329

FastFaxts 1747-002

386 DEVELOPMENT TOOLS

	List	Ours
386 Max	75	69
DESQview 386	220	189
FoxBASE+/386	595	479
High C - 386	895	875
Hoops exted. DOS MetaWare	795	759
VM/386	245	229

AI EXPERT SYSTEMS

Exsys Professional	795	695
Logic Gem by Sterling Castle	99	89
VP Expert	249	219

AI-LANGUAGES

ARITY Combination Package	1095	989
PC Scheme LISP- by TI	95	85
TURBO PROLOG V2.0	150	119

ASSEMBLERS

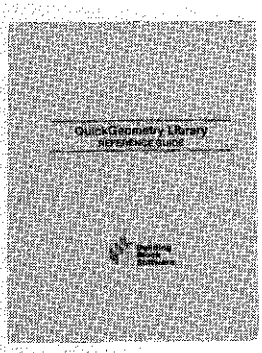
386 IASM/LINK by Pharlap	495	Call
MS Macro Asm	150	105
Turbo Assembler/Debugger	150	109

C LANGUAGE COMPILERS

High C-286 - by MetaWare	595	590
High C-386 - by MetaWare	895	875
Instant C/16M	795	769
Lattice C - 6.0	250	189
Microsoft C5.1 - w/CodeView	450	299
Micorsoft QuickC	99	69
w/serial mouse	199	137
Turbo C - by Borland	150	Call
WATCOM C7.0	395	329
WATCOM C7.0 386	895	799
Zortech C	Call	Call

CASE & PROTOTYPERS

Dan Bricklin Demo II	195	186
Interactive EasyFlow	150	135
Instant Replay - Nostradamus	150	139
Matrix Layout - flow chart	200	179
MetaDesign by Meta Software	350	295
Pro-C	495	429
ProtoFinish by Genesis	300	279
R&R Code Generator	149	129
Show Partner F/X	395	279



QuickGeometry Library V1.1
by Building Block Software

Write the CAD/CAM software you need! Use QuickGeometry Library to develope: EE and ME parametric programs, NC post processors, motion control software, FEA mesh generators, entire CAD/CAm systems-without spending any time on math routines. QGL C routines increase number crunching speed by more than 100x over AutioLISP. Supports: points, vectors, lines, arcs, circles, ellipses, B-splines(NURBS), rotation, translation, scaling, mirroring, parallel offsetting, intersection, trimming, reading and writing DXF files, VGA, EGA, CGA, Hercules geometry display; lists.INcludes C source for Microsoft C and Turbo C. No royalties
List: $199 Ours: $179

FastFaxts 1714-001

Visible Computer 80286	100	85

BASIC & ADD-ONS

BAS-C Commercial	495	439
dB/Lib	139	125
MS QuickBASIC V4.5	99	69
ProBas Lib by Hammerly	159	Call
ProRef by Hammerly	50	Call
ProScreen by Hammerly	99	Call
ProMath by Hammerly	99	95
QBase	149	139
QuickPak Professional	149	139
Softcode/w Template	100	89

BOOKS/VIDEOS

Mouse Programmer's Reference	30	24
Programming Windows	25	19
Running MS-DOS, 4th Edition	23	18
Zortech C Video Course	300	Call
Zortech C++ Video Course	Call	Call

Visable Analyst	595	585

COBOL

MS COBOL V3.0	900	Call
Realia COBOL	995	859

COMMUNICATIONS ADD-ONS

C Asynch Lib	249	219
C Asynch Manager 3.0	189	155
Essential Lib by S. Mountain	329	309
Greenleaf Comm Library	299	249
QuickComm	139	129

DATABASE MANAGEMENT

Cause by Maxem	295	259
CLARION (Professional Developer V2.1)	845	549
dbFast Windows	395	329
D the data language	395	359
Magic PC	499	479
PC Plus -Texas Instruments	1995	1799
Paradox V3.0	725	479
R-BASE for DOS	725	499
Tas Lan Support	200	185
Tas Professional	499	459

DBASE

Clipper Summer '87 MU	695	459
db/Fast	99	89
dBASE IV	795	Call
dbFAST Windows	395	329
dbFAST/PLUS	345	319
dBMAN V	295	275
Force by Sophco	695	395
FoxPro	795	495
FoxBASE + - V2.1	395	279
QuickSilver Diamond	599	399
Sunshow Professional	100	95

DBASE ADD-ONS

AdComm For Clipper	295	279
Artful Lib	200	Call
CodeBASE 4	295	279
dGE - business graphics	195	179

Knowledge Pro's Toolkit	129	119
R&R Relational Reportwriter	149	139
Scrimmage by Synergy - screen/menu	99	89
Sycero db - single user	495	479
UI Programmer - Dev's Version 2.0	595	479

DEBUGGERS/ DISASSEMBLERS

Dis Doc	130	119
Periscope II & III - breakout switch	Varies	Call
RE:Source by Genesoft	150	119
SoftProbe	395	345
Sourcer	100	89
sPORT by WHS	80	Call
Trapper	200	189
Zortech C Debugger	90	Call
Zortech C++ Debugger	150	Call

DEVELOPMENT TOOLS

AsmFlow	100	89

The BEST linker for MS-DOS.....

.RTLink/Plus *by Pocket Soft*

The only MS-DOS linker with CodeView support for multiple/nested overlays. Advanced technology allows more overlays, overlay areas, symbols, objects, classes, and sections than any other linker. Smart caching of most recently accessed overlay sections in conventional, expanded, and extended memory (New XMS specifications). Easy to use Profiler provides detailed performance statistics without source code modification. Run-Time Libraries (RTLs) speed development time and reduce disk-space/modem-time requirements. Source code provided for overlay manager and Profiler's read-analysis utility. LIBLIST provides detailed information on inter-relationships between modules-great for optimizing and designing overlay schemes. Highest score of all linkers reviewed by *PC Week* 1/29/90.
List: $495 Ours: $419

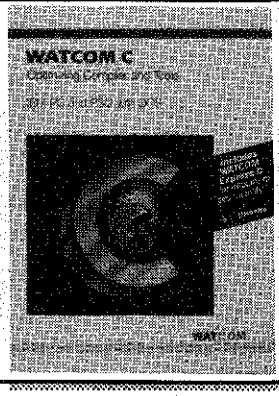
FastFaxts 1277-003

dQuery MU	495	469
FLIPPER Graphics Library	195	179
FUNCcy.LIB	195	179
Pro Clip	149	125
R&R Relational Reportwriter & Clipper module	165	149
SilverComm Library	189	169
SilverPaint	100	89
Steve Straley's Toolkit	180	149
Tom Rettig's Library	100	85

DBASE TOOLS

Buzzwords dAnalyst	295	269
CLEAR + for dBASE	200	179
dAnalyst	295	269
Dbase Online - 6 pop-up refs.	149	129
BRIEF w/dBRIEF	285	Call
dBug - source debugger	195	179
dBX - dBASE III to C	550	529
dSalvage	100	89
Genifer - code generator	395	269

Autoflow C/Pascal	299	279
C Programmers Toolbox	99	89
CLEAR + for C	200	169
Codan	395	349
dAnalyst for Clipper	295	269
Documenter	295	245
Inside	125	119
Knowledge Pro	495	419
Knowledge Pro Windows	695	589
MKS Lex & Yacc	249	199
MKS RCS	189	175
Poly Doc-SU	199	179
PC-Lint	139	115
Plink 86 Plus - overlays	495	399
PolyMake	149	135
PVCS Personal	149	135
PVCS Professional	695	659
.RTLINK - by Pocket Soft	295	279
.RTLINK Plus	495	419
Sbrowse	245	Call
Source Print by Powerline	99	Call



WATCOM C8.0
by WATCOM

WATCOM C8.0/386 is 100% ANSI C optimizing compiler/runtime library for Intel's 80386 architecture, generating applications for 32-bit protect mode. Features include: protected mode version of the compiler; VIDEO full-screen source-level debugger; MS library-& source- compatibility; execution profiler;high proformance linker; graphics library;supports MetaWare High C 386 runtime calling conventions; SAA compatible.
WATCOM C8.0 for 16-bit 80X86 DOS systems & OS/2 . features include: protected mode version of the compiler, Windows support, OS/2 support: MS library & source compatibility; execution profiler; remote debugging capability; SAA compatible
List: $1295 (386) Ours: Call
List: $ 495 Ours: Call

FastFaxts 1044-004

THE PROGRAMMER'S

Call today for complete technical information and advice, our free catalog, literature and *solid value.* ***Prices subject to change without not***

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

For Professional Developers!

Sourcer w/BIOS Pre-ProC	140	129
TLIB	100	89
Zortech C++ Tools	150	Call

EDITORS

BRIEF	199	Call
Edix	195	179
Epsilon - like EMACS	195	169
KEDIT - like XEDIT. V/4	150	139
SPF/PC - V2.1	245	199
Vedit Plus	185	139

FILE ADD-ONS

Btrieve ISAM - V5.0	245	199
Btrieve/N - multiuser	595	479
Xtrieve + - by Novell	595	479
C-Data Manager	150	139
c-tree by Faircom - source	395	329
d-tree - data dictionary	495	459
r-tree - report generator	295	245
c-tree - w/r-tree	650	549
CBTREE - Source	195	179
CQL - full w/ source	395	359
dBC ISAM III - by Lattice	250	229
db_FILE - network model	Varies	Call
db_RETRIEVE - SQL	Varies	Call
XQL	795	649
WKS Library	195	189

FORTRAN

FOR_C - FORTRAN '77 w/so	875	789
Fortran Opt. Compiler	450	309
Lahey FORTRAN F77L	595	549
Lahey Personal FORTRAN	95	Call
MS FORTRAN - CodeView	450	309
Spindrift - DOS, windows	149	129

GENERAL ADD-ONS

C + O Class Library	195	179
C Tools Plus - V6.0	149	109
C Utilities-by South Mountain	185	137
Greenleaf Functions	209	189
Greenleaf SuperFunctions	299	239
Turbo C Tools by Blaise	149	134

GRAPHIC ADD-ONS

Codemaster II	289	269
Essential Graphics	399	349
GraphiC	395	319
Greenleaf Makeform-DOS	125	119
GSS Dev't Toolkit	595	525
Halo '88 - 140 + devices	395	279
Hoops	795	759
MetaWINDOW	250	209
MetaWINDOW/PLUS	325	289
PCX Programmer's Toolkit	195	179
Pizazz Plus	149	129
QuickWindows Advanced	139	129
Sunscan	250	229
Sunshow Adv Image Toolkit	250	239
Sunshow C Image	250	229

NETWORKS

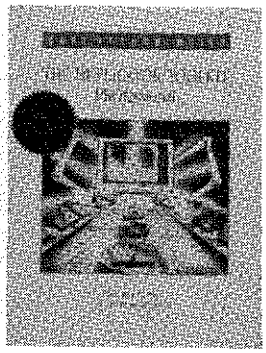
Netback	245	229
NETmanager	1495	1295
cc:Mail 25 user	695	649
DaVinci eMAIL	995	895
Netware SQL	595	519
Netware C Interface	295	239

OBJECT-ORIENTED

Actor	695	639
C Talk	150	139
C Talk/views	450	419
C++ 80386 by Intek	495	469
Guidelines C++ V2.0	395	379
Smalltalk/V	100	85
Zortech C ++	200	Call

OS - SUPPORT

DESQview	130	109
MKS Toolkit - Unix shell	249	229
MS Windows/286	99	69
MS Windows/386	195	139
Development Toolkit/Wind	500	319
Omni View	80	69
OS/286 by Eclipse	495	459
OS/386 by Eclipse	495	459



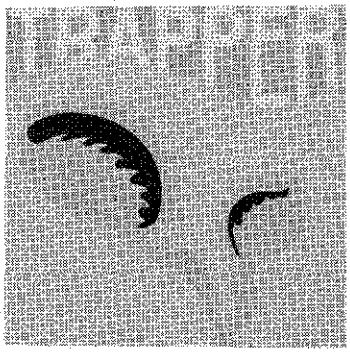
The Duplicator Toolkit Pro 3.0

by Copy Technologies

End The Flopy Shuffle! This super fast diskette duplication utility copies a 360K floppy in 20 seconds. Format, copy and verify "on the fly" in one keystroke. Make serialization labels in background. Stores your master image in RAM or permanently on the hard drive. Supports 360K, 720K, 1.44M drives. Alternates "A" "B" for even faster copies.

List: $129 Ours:$119

FastFaxts 1419-003



TRAPPER

by Purart, Inc.

HARDWARE DEBUGGER FOR BORLAND'S TURBO DEBUGGER

- Watches memory and I/O locations for read/writes.
- Breakout button interrupts runaway programs.
- Set breakpoints in ROM.

Trapper is a half card hardware debugger board designed especially for Borland's Turbo Debugger. Trapper ads to Turbo Debugger with hardware "watch and stop" capability to easily find memory bashing bugs. Programs run at ful speed until errors occur.

IBM PC/XT or PC/AT

List: $200 Ours: $189

FastFaxts 1528-001

OTHER LANGUAGES

Modula-2 Dev. System- by Logitech	249	229
RPG II Dev. Systems	1600	1469
TopSpeed MODULA-2 Compiler	100	89
Turbo Prolog V2.0	150	119

OTHER PRODUCTS

386 Max Professional	130	119
Carbon Copy Plus	199	159
CO/SESSION	249	229
Dan Bricklin's PageGarden	100	89
The Duplicator Toolkit-Pro 3.0	129	119
Fastback Plus	189	159
File Shuttle by GETC	120	109
Flow Charting II Plus	229	179
FM-Plus	100	89
Headroom by Helix	130	89
HiJaak	149	139
Hyperpad by Brightbill	149	129

TURBO PASCAL

graphics-Menu	149	139
Say What	50	Call
Topaz	75	69
TP2C	249	229
Turbo Analyst	99	89
Turbo ASYNCH PLUS	189	169
Turbo Pascal 5.0 by Borland	150	109
Turbo POWER TOOLS PLUS	149	109
Turbo Professional	125	109
Turbo Programmer	300	269

TEXT SCREEN ADD-ONS

AEWindows	225	209
C Windows Toolkit	100	89
C Worthy w/forms	295	Call
Greenleaf DataWindows	395	339
Hi-Screen XL	149	129
Hi-Screen XL Professional	325	289
Hyperinterface- TFN, C, PAS menus	100	95

Clarion Professional Dev 2.1

by Clarion Software

Professional Developer 2.1 is an immensely powerful, easy-to-use DBMS application developer, capable of cutting applications development time by 50%. Allows import-export of dBASE BASIC and DIF files; interfaces with routines from C and Assembler. Applications generator for creating screens, menus, windows, reports, and tables. Includes Report Writer for creation of ad hoc reports and queries. Comes with built-in LAN support, with no run-time system required for distribution. Recent winner of *PCWeek* poll of corporations using programmable databases.

List: $845 Ours: $549

FastFaxts 1005-004

LapLink III	150	129
Link & Locate ++ - ROM MSC	395	349
Math Advantage	495	475
Mirror II	70	59
Net Lib	249	229
Norton Commander	149	119
Norton Utilities Advanced	150	109
Paginate by Accumatics	100	89
PC Anywhere III	145	129
PC Tools Deluxe	129	109
PC Kwik Power Pack	130	119
QEMM-386	100	89
Remote2	195	139
Resident C by South Mountain w/source	99	85
Service Diagnostic Kit	249	199
SMX (College)	195	Call
SpinRite II	95	Call
Time $heet	89	Call
Coterm 220	150	135
JAM by JYACC	245	219
MEWEL	595	469
POWERSCREEN by Blaise	179	149
Vitamin C - source, menus	149	109
VC Screen - painter	225	169
Vermont Views-Obj+Source	149	139
Zortech Pro Screen	890	819
	70	Call

UNIX/XENIX

Application Devel. 2 user	1445	1349
Edix - editor	425	399
ESIX/V 386 Devlp (2 user)	595	569
Guidelines C++ for V/386 V2.0	495	479
Informix SQL	995	929
JAM JYACC Applic. Mgr.	many	Call
NDP Fortran-386	795	749
Network Developer 2 user	1495	1399
Vermont Views	1595	1349
Workstation Devel. 2 user	1995	1869
Workstation Starter 2 user	495	469
XENIX 286 Complete System	1495	1279
XENIX 386 Complete System	1595	1369

FastFaxts service!

Instant product literature via fax 24 hours a day!
Call 617-740-0025, from your fax machine, and enter FastFaxts code. (See Catalog)

Free Catalog!

Over 1,750 development tools listed. Call for your free copy.

Mention "PJ590"

Credit Card orders processed only when product is shipped.

SHOP • 800-421-8006

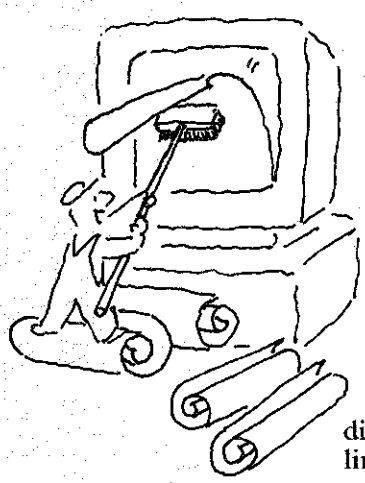
5 Pond Park Road, Hingham, MA 02043 • Canada 800-446-3846 • MA 617-740-2510 • FAX: 617-749-2018

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Two great tools.

NEW VERSION 2.8
CLOCK & EDIT UNITS, DIALOG BOXES, AND PROGRESS BARS!

SAYWHAT?!
The lightning-fast screen generator.



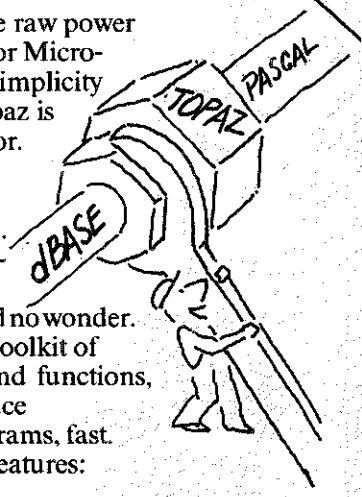
Whether you're a novice programmer longing for simplicity, or a seasoned pro searching for higher productivity, you owe it to yourself to check out Saywhat. You see, with Saywhat, you can build beautiful, elaborate, color-coded screens in minutes! That's right. Truly *fantastic* screens for menus, data entry, data display, and help-panels that can be displayed with as little as one line of code in *any* language.

Here's what you get:

- Design screens, windows, and *moving bar menus!*
- Easy-to-use, powerful editor lets you create screens in a jiffy.
- Pop up your screens and menus with one line of code in dBASE, all the dBASE compilers, your favorite BASIC, Pascal, *or any other language!*
- Screen Library Manager.
- Generates runtime code.
- No runtime license or royalty fees.
- Comes with a 100 page manual, plus dozens of sample programs and free utilities.

$49.95

TOPAZ
The breakthrough DBMS toolkit for Pascal



If you'd like to combine the raw power and speed of Turbo Pascal or Microsoft's QuickPascal with the simplicity and elegance of dBASE, Topaz is just what you're looking for. That's because Topaz was specially created to let you enjoy the best of *both* worlds. The result? You create complete, truly dazzling applications in a very short time. And no wonder. Topaz is a comprehensive toolkit of dBASE-like commands and functions, designed to help you produce outstanding, polished programs, fast. Check out these powerful features:

- Over 220 routines all with easy-to-use, dBASE-like syntax.
- Data entry routines like **SAY, GET, PICTURE, RANGE,** color selection, unlimited data validation.
- Open up to 10 DBF files, with up to 15 indexes with **USE, SELECT, SKIP, APPEND, PACK, INDEX ON, SET INDEX TO,** and **FIND.**
- No need to buy dBASE. **CREATE, BROWSE** and **REPORT** utilities included.
- Easily implement Saywhat and Lotus-style moving bar menus.
- **BROWSE or EDIT** any DBF file with just one line of code! Programmable and windowed too.
- Pick from windowed data or filenames with one line of code.
- Comprehensive Time & Date math in 7 international formats.
- Powerful code and report generators included!
- Comes with a complete 320 page manual, plus sample programs to get you started.

$74.95

ORDER NOW.
YOU RISK NOTHING.

Thousands of satisfied customers have already ordered from us. Why not visit your dealer or call toll-free, right now and put Saywhat and Topaz to the test yourself? They're fully guaranteed. You don't risk a penny.

Special limited-time offer! Save $26. Buy Saywhat?! and Topaz together for just $99 (plus $5 shipping and handling, $10 Canada, $25 International, Calif. residents add 7%).

Visit your nearest dealer
or call toll-free:
800-468-9273
In California: 800-231-7849
International: 415-571-5019

Software Science, Inc.
100 Valley Drive, Brisbane, CA 94005

MONEY BACK GUARANTEE.
If you aren't completely delighted with Saywhat or Topaz, for any reason, return them within 30 days for a prompt, friendly refund.

Dealers: SAYWHAT?! and TOPAZ are available from Kenfil Distribution, and in Europe from ComFood Software, W. Germany 49-2534-7093

Guaranteed!

SOFTWARE SCIENCE INC.

Number **198** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Reusing objects—it's right up OOP's alley

Let's list again, and again, . . .

By Tom Swan

I'm crazy about lists. I collect to-do lists, shopping lists, and job lists. I've crafted book-idea lists longer than short novels. I hoard well-pruned lists of Christmas cards to send, books to read, movies to watch, and friends to call. Once, before I realized I was going too far, I listed all my lists so I could keep track of them. I probably have more lists than the CIA.

So, when I wrote the disk-directory module for my Turbo Pascal 5.5 program VA (for "View Anything"), I naturally turned to the list object I had written months ago. After all, a directory is just a list of filenames. And that's what OOP is supposed to be good for—building reusable object types (or classes) to use in new and unplanned ways.

While writing the code, I made two discoveries. One, OOP *does* encourage building new object types from ancestors. Call me a skeptic, but I'm accustomed to what happens with most theories, which often wash out like old socks. Sooner or later, the weak points disintegrate into holes. But reusability is one theory that doesn't seem to require darning. I was able to reuse my old objects with no changes. Not one.

The second discovery I made was more surprising. Instead of the specialized directory object I had planned, I ended with a new general-purpose list-selector object for my growing OOP library. I didn't plan this object—it "just happened." And, as a result, I learned an important rule that's helping me to create other reusable object types. But more on that in a moment—first, let's bring newcomers up to date.

REPRISE

In the previous two installments [see *Programmer's Journal* volumes 7.6 (November/December 1989) and 8.1 (January/February 1990)], I described some of the raw materials of VA, a program that will let me view disk files of different kinds—graphics, text, spreadsheets, and others. In this article, I'll present more of my VA prototype. I plan to share VA's source code in *PJ* while taking a peek (and an occasional poke) at OOP techniques.

You should be able to understand the high points even if you've joined us in progress, but to compile the listings, you'll

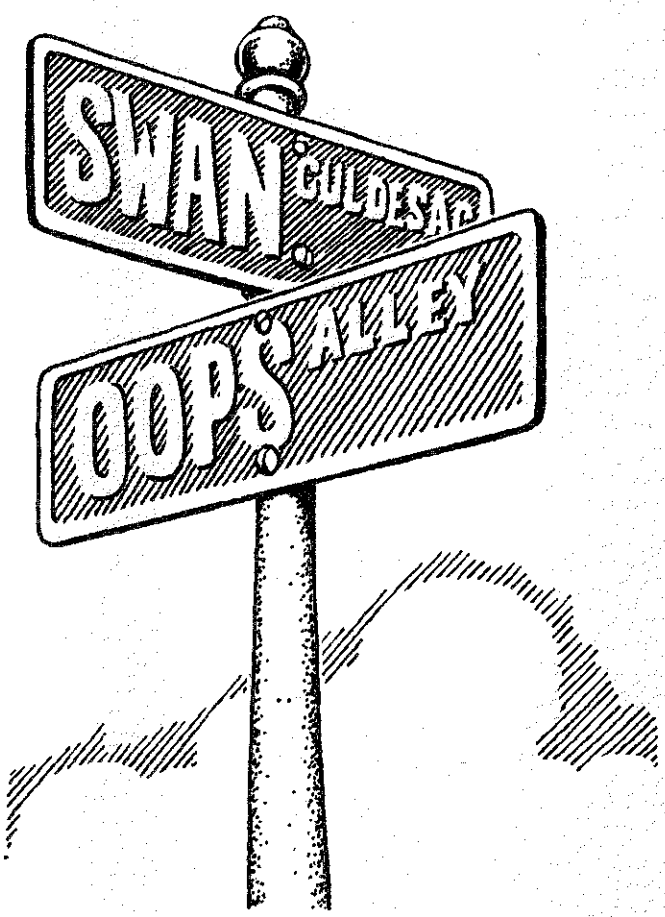


ILLUSTRATION BY JIM CARPENTER

Tom Swan is a contributing editor of Programmer's Journal *and the author of* Mastering Turbo Pascal 5.5 *and* Mastering Turbo Assembler, *both published in 1989 by Howard W. Sams, Indianapolis, Indiana. Send mail to Tom in care of* PJ, *or to his CompuServe ID: 73627,3241.*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Objects should be reusable. But designing objects that are isn't so automatic as is often decreed.

need the files from previous months. If you purchase the diskette for this issue, you'll find the missing files in VA.ARC.

A SELECT EXAMPLE

Oodles of real and imagined OOP experts have been telling the world of OOP's strengths—and "reusability" is their favorite new buzzword. I sometimes wonder how many objects these writers have reused lately, but they have a point. Objects *should be* reusable. But designing objects that *are* isn't so automatic as is often decreed.

Concentrating on writing practical code helps—objects worth reusing should be usable from the start, or, *first, make it usable*. If you construct every object that comes to mind and carve in every imaginable ornament, your object hierarchy will soon resemble a royal family tree. Better to construct an object as you would a chair, not a throne. Don't try to conquer the continent; be satisfied with a comfortable fit.

As an example of the "make it usable" rule, Listing 1 shows how my plans changed over time. At first, I had **fileItem** and **directory** descend from the **strItem** and **list** objects from my class library. But I soon realized I had bitten off too much. Having grown like a bamboo thicket, my old UDIR.PAS was impenetrable.

So I began again. As I looked at my wild crop of code, I saw I had created, not one, but two object classes—one to list filenames from a disk directory and another to select any kind of object from a list. After hacking through the undergrowth, I uncovered the objects needed to read directories. The first such object descends from **selectableItem**, which I had not yet created:

```
fileItemPtr = ^fileItem;
fileItem = object( selectableItem )
   constructor init( sr : searchRec );
end; { fileItem }
```

FileItem contains only one method, a constructor **init**, which initializes a variable (or *object instance*) of this type, using a DOS unit **searchRec** to read filenames from disk. Despite its

LISTING 1

```
(* udir.pas -- (c) 1990 by Tom Swan *)

unit udir;
interface
uses dos, ulist, uselect;

type

   fileNameStr = string[12];      { e.g. '*.PAS' or 'MYFILE.TXT' }

   fileItemPtr = ^fileItem;
   fileItem = object( selectableItem )
      constructor init( sr : searchRec );
   end; { fileItem }

   directory = object( selectionList )
      sr : searchRec;    { From the DOS unit }
      wildCard : fileNameStr;
      constructor init( wc : fileNameStr; x1, y1, x2, y2 : word );
      procedure resetDirectory; virtual;
      procedure putWildCard( wc : fileNameStr );
   end; { directory }

implementation

{ ----- Initialize fileItem object. }
constructor fileItem.init( sr : searchRec );
var
   junk : dirStr;       { Required by fSplit(); ignored }
   fileName : nameStr;  { File name without extension }
   fileExt : extStr;    { File name extension with leading '.' }
begin
   fSplit( sr.name, junk, fileName, fileExt ); { Parse file name }
   selectableItem.init( fileName + fileExt, TRUE ) { Init ancestor }
end; { fileItem.init }

{ ----- Initialize directory object. }
constructor directory.init( wc : fileNameStr; x1, y1, x2, y2 : word );
begin
   selectionList.init( x1, y1, x2, y2 );
   putWildCard( wc )   { Assign wild card and read directory }
end; { directory.init }

{ ----- Reread current directory. }
procedure directory.resetDirectory;
begin
   disposeList;   { Dispose an existing directory list }
   findFirst( wildCard, 0, sr );
   while dosError = 0 do
   begin
      insertItem( new( fileItemPtr, init( sr ) ) );
      findNext( sr )
   end
end; { directory.resetDirectory }

{ ----- Change wild card specification. }
procedure directory.putWildCard( wc : fileNameStr );
begin
   wildCard := wc;
   resetDirectory
end; { directory.putWildCard }

end. { udir }
```

LISTING 2

```
(* uselect.pas -- (c) 1990 by Tom Swan *)

unit uselect;
interface
uses crt, ukeys, uscreen, uitem, ustritem, ulist;

type

   keySet = set of char;

   selectableItemPtr = ^selectableItem;
   selectableItem = object( strItem )
      selected : Boolean;
      constructor init( s : string; initial : Boolean );
      procedure setSelection( setting : Boolean );
      procedure toggleSelection;
      function getMarkedString : string; virtual;
   end; { selectableItem }

   selectionListPtr = ^selectionList;
   selectionList = object( list )
      xmin, ymin, xmax, ymax : word;   { Display coordinates }
      constructor init( x1, y1, x2, y2 : word );
      procedure firstSelectedItem( var noItems : Boolean );
      procedure nextSelectedItem( var pastHead : Boolean );
      procedure setAll( setting : Boolean );
      procedure toggleAll;
      procedure displayItems; virtual;
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

apparent simplicity, **fileItem** inherits all of its ancestor's data fields (or *instance variables*) and methods. Because I planned to have **selectableItem** inherit from yet another object, **strItem**, I knew that **fileItem** had all the tools needed to save, retrieve, and list filename strings.

I also needed an object to hold a list of **fileItem**s. Originally, **directory** was descended from **list**, but I changed the ancestor to another nonexisting object, **selectionList**:

```
directory = object( selectionList )
   sr : searchRec;   { From the DOS unit }
   wildCard : fileNameStr;
   constructor init( wc : fileNameStr;
      x1, y1, x2, y2 : word );
   procedure resetDirectory; virtual;
   procedure putWildCard( wc : fileNameStr );
end; { directory }
```

The **searchRec** field **sr** holds the information required to initialize new **fileItem** objects by calling that object's **init** constructor. The **wildCard** string holds a wildcard, for example, *.PAS or PART00?.*.

Constructor **init** assigns a string argument **wc** to **wildCard** and calls the ancestor's constructor to initialize the object. The other parameters are display coordinates for **selectionList**.

Two other methods reload the directory from disk (**resetDirectory**) and change the current wildcard (**putWildCard**). The first of these is virtual, causing TP to add the method's address to the object's VMT (virtual-method table), through which method calls are redirected. The second method is static—calls to it are no different than to other procedures and functions.

I used a simple rule of thumb to decide which of those methods to make virtual and which static. **PutWildCard** calls **resetDirectory**. If I have to replace **resetDirectory** in a future descendant, that method *must* be virtual to allow the already compiled **putWildCard** to call the replacement. But **putWildCard**, which merely assigns a new **wildCard** value, may as well be static—it's unlikely to require future changes.

When you read through Listing 1, notice the simple method implementations. Three of them contain only two statements each. This is a mark of good OOP technique: short methods that do the job and end. If your methods are longer than about 25 or 30 lines, consider redesigning.

THE SELECT FEW

Listing 2, USELECT.PAS, contains the cuttings from my original overgrown UDIR.PAS. After I realized that selecting objects from a list was a general-purpose task that I could use for many jobs, the unit almost wrote itself.

```
      procedure selectItems( exitKeys : keySet;
         var lastKey : char ); virtual;
   end; { selectionList }

implementation

const

   DIM      = FALSE;    { ShowItem arguments }
   BRIGHT   = TRUE;
   ENTER    = #13;      { Various character constants }
   ESC      = #27;
   BLANK    = #32;
   KEY_UP   = 'H';
   KEY_DOWN = 'P';

{ ----- Initialize selectableItem object. }
constructor selectableItem.init( s : string; initial : Boolean );
begin
   selected := initial;
   strItem.Init( s )
end; { selectableItem.init }

{ ----- Set selected switch to 'setting'. }
procedure selectableItem.setSelection( setting : Boolean );
begin
   selected := setting
end; { selectableItem.setSelection }

{ ----- Toggle this item on/off }
procedure selectableItem.toggleSelection;
begin
   selected := not selected
end; { selectableItem.toggleSelection }

{ ----- Return string with selected marker on or off }
function selectableItem.getMarkedString : string;
var
   s2 : string[2];
begin
   if selected
      then s2 := '> '
      else s2 := '  ';
   getMarkedString := s2 + getString
end; { selectableItem.getMarkedString }

{ ----- Initialize selectionList object }
constructor selectionList.init( x1, y1, x2, y2 : word );
begin
   list.init;
   xmin := x1;
   ymin := y1;
   xmax := x2;
   ymax := y2
end; { selectionList.init }

{ ----- Advance list to first item where selected=TRUE. If noItems is
TRUE, then the list contains no selected items. }
procedure selectionList.firstSelectedItem( var noItems : Boolean );
begin
   noItems := listEmpty;   { i.e. noItems = TRUE if list is empty }
   if noItems then exit;
   resetList;
   if selectableItemPtr( currentItem )^.selected then exit;
   nextSelectedItem( noItems )
end; { selectionList.firstSelectedItem }

{ ----- Advance list to next item where selected=TRUE. If pastHead
returns TRUE, then the search has gone full circle around the list. }
procedure selectionList.nextSelectedItem( var pastHead : Boolean );
begin
   pastHead := listEmpty;
   while not pastHead do
   begin
      nextItem;
      pastHead := atHeadOfList;
      if selectableItemPtr( currentItem )^.selected then exit
   end { while }
end; { selectionList.nextSelectedItem }

{ ----- Set all select switches to 'setting.' }
procedure selectionList.setAll( setting : Boolean );
begin
   if listEmpty then exit;
   resetList;
   repeat
      selectableItemPtr( currentItem )^.setSelection( setting );
      nextItem
   until atHeadOfList
end; { selectionList.setAll }

{ ----- Toggle all select switches for all listed items. }
procedure selectionList.toggleAll;
begin
   if listEmpty then exit;
   resetList;
   repeat
      selectableItemPtr( currentItem )^.toggleSelection;
      nextItem
   until atHeadOfList
end; { selectionList.toggleAll }
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Look for similar relationships. . . . Exploit subtle differences. . . . Don't rebuild—reuse.

I decided to create two objects—one for the selectable items and one for a list of those items. Although I could have created the first of these, **selectableItem**, as a descendant of **item**, I decided to use **strItem** instead. Most of my lists will contain string objects anyway, and so this seemed the best choice.

```
selectableItemPtr = ^selectableItem;
selectableItem = object( strItem )
   selected : Boolean;
   constructor init( s : string;
      initial : Boolean );
   procedure setSelection( setting : Boolean );
   procedure toggleSelection;
   function getMarkedString : string; virtual;
end; { selectableItem }
```

SelectableItem is a **strItem** with a single Boolean field, **selected**, that indicates whether the object is among the chosen few. In other words, a **selectableItem** is just a string item that can be selected. When building your own object hierarchies, look for similar relationships between objects Charlie and Sam, who could be identical twins, except that Charlie has a beard. Exploit subtle differences among objects. Don't rebuild—*reuse*.

The object's constructor requires two parameters: a string **s** for passing to **strItem**'s constructor, and **initial** to indicate whether to select a newly created object. For example, if **merchandise** is a variable of type **selectableItem**, this assigns a string to **merchandise** and selects the object:

```
merchandise.Init( 'Jacket', TRUE );
```

If **merchandisePtr** is a variable of type **selectableItemPtr**, initialize a new object on the heap and assign its address to **merchandisePtr** by calling TP's extended **new** procedure, passing the constructor as the second argument:

```
new( merchandisePtr, Init( 'Coat', FALSE ) );
```

```
{ ----- Display all selectableItem objects }
procedure selectionList.displayItems;
var
   line : integer;
begin
   clearWindow( xmin, ymin, xmax, ymax );
   gotoxy( xmin, ymin );
   line := ymin;
   if listEmpty then exit;    { Leaving window clear }
   resetList;
   repeat
      gotoxy( xmin, line );
      write( selectableItemPtr( currentItem )^.getMarkedString );
      nextItem;
      inc( line )
   until atHeadOfList or ( line > ymax )
end; { selectionList.displayItems }

{ ----- Let user select one or more items from the list. }
procedure selectionList.selectItems(
   exitKeys : keySet; var lastKey : char );
var
   line : integer;            { Line number (absolute) }
   exitKeyPressed : Boolean;  { True if any exit key pressed }

   procedure showItem( ip : ItemPtr; highlight : Boolean );
   begin
      gotoxy( xmin, line );
      clrEol;
      if highlight
         then highVideo
         else normVideo;
      if ip <> nil
         then write( selectableItemPtr( ip )^.getMarkedString );
      normVideo
   end; { showItem }

   procedure startList;
   begin
      displayItems;
      line := ymin;
      resetList;
      showItem( currentItem, BRIGHT )
   end; { startList }

   procedure moveUp;
   begin
      if (not listEmpty) and (not atHeadOfList) then
      begin
         showItem( currentItem, DIM );
         if line > ymin
            then dec( line )
            else scrollDown( xmin, ymin, xmax, ymax );
         prevItem;
         showItem( currentItem, BRIGHT )
      end { if }
   end; { moveUp }

   procedure moveDown;
   begin
      if (not listEmpty) and (not atEndOfList) then
      begin
         showItem( currentItem, DIM );
         if line < ymax
            then inc( line )
            else scrollUp( xmin, ymin, xmax, ymax );
         nextItem;
         showItem( currentItem, BRIGHT )
      end { if }
   end; { moveDown }

   procedure toggle;
   var
      sip : selectableItemPtr;
   begin
      sip := selectableItemPtr( currentItem );
      if sip <> nil
         then sip^.toggleSelection;
      showItem( sip, BRIGHT )
   end; { toggle }

begin
   startList;
   exitKeys := exitKeys + [ ESC ];    { Escape always exits }
   exitKeyPressed := FALSE;
   repeat
      lastKey := upcase( getKey );
      case lastKey of
         KEY_UP      : moveUp;
         KEY_DOWN    : moveDown;
         BLANK,ENTER : toggle;
      else
         exitKeyPressed := lastKey in exitKeys
      end
   until exitKeyPressed;
   showItem( currentItem, DIM )
end; { selectionList.selectItems }

end. { uselect }
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HARVARD SOFTWORKS

NUMBER ONE IN FORTH INNOVATION

(513) 748-0390 P.O. Box 69, Springboro, OH 45066

MEET THAT DEADLINE ! ! !

- Combine the raw power of extensible languages with the convenience of carefully implemented functions!
- Execute faster than optimized C!
- Compile 40,000 lines per minute!
- Stay totally interactive, even while compiling!
- Program at any level of abstraction from machine code thru application specific language with equal ease and efficiency!
- Modify routines without recompiling!
- Use source code for 2500 functions!
- Use data structures, control structures, and interface protocols from any other language!
- Implement that borrowed feature, often more efficiently than in its source!
- Use an architecture that supports very small programs or full megabyte ones with a single version!
- Forget chaotic syntax requirements!
- Use subroutine libraries written for other languages! More efficiently!
- Outproduce experienced programmers stuck using conventional languages!

HS/FORTH with FOOPS - The only full multiple inheritance object oriented language under MSDOS!

WAKE UP ! ! !

Forth is no longer a language that tempts programmers with "great expectations", then frustrates them with the need to reinvent simple tools expected in any commercial language.

HS/FORTH Meets Your Needs!

We don't shortchange you with promises. We provide implemented functions to help you complete your application quickly.

Don't judge Forth by public domain products or ones from vendors primarily interested in consulting - they profit from not providing needed tools! Public domain versions are cheap - if your time is worthless. Useful only in learning Forth's basics, they fail at showing its true potential. Not to mention being s-l-o-w.

You can't add extensibility to fossilized compilers. You can easily add features from any other language to **HS/FORTH**. Try our full modifiable multiple inheritance **OOPs** extension or our **Fortran** translator & mathpak!

HS/FORTH has more features and libraries than we can possibly describe here. **Please ask for our brochure!**

HS/FORTH uses MSDOS or PCDOS. Each level includes features of lower ones. Level upgrades: $25. plus price difference. Sources in ordinary text files.

STUDENT LEVEL	**$145.**
text & scaled/clipped graphic windows, ellipses, splines, arcs, fills, turtles; powerful parsing, formatting, file and device I/O; shells; interrupt handlers	
PERSONAL LEVEL	**$245.**
software floating point, trig, transcendental, quad & scaled integer, automatic optimizer for assembly like performance.	
PROFESSIONAL LEVEL	**$395.**
hardware floating point, turnkey, seal, Rosetta Stone Dynamic Linker, round robin & time slice multitaskers, dynamic string manager, file blocks, sector mapped blocks, assemblers	
PRODUCTION LEVEL	**$495.**
Metacompiler: DOS/ROM/direct/indirect C data structures & struct compiler, TurboWindow-C graphics library,	

Options:

FOOPS+ with multiple inheritance	**$ 75.**
286FORTH or **386FORTH**	**$295.**
TurboWindow-C (Metagraphics)	**$ 99.**
BTRIEVE for HS/FORTH (Novell)	**$199.**
ROMULUS HS/FORTH from ROM	**$ 95.**
FFORTRAN translator/mathpak	**$ 75.**

Number 42 on the Reader Service Card

Computer Professional's Legal Toolkit

PROBLEM:
Almost 100% of warranty disclaimers will not stand up in court!
CONSEQUENCES:
Almost Unlimited Personal Liability For User Losses!
SOLUTION:
THIS PROGRAM!

Only $159.95
Order Now!
Call 24 hours
Most cards accepted!

1-800-648-1300

R. Fringe Publishers, Inc.
Box 796, Casselberry FL 32707

FOR PROGRAMMERS, VARS, CONSULTANTS, RETAILERS, DEVELOPERS-HARDWARE/PERIPHERAL MFGS - ANY COMPUTER ENTREPRENEURS

Full Annotated Text of Laws, Court Cases and Government Regs.

Mini-Seminars

Legal Document Generator using A.I. with more than 40 forms that are valid in all 50 states

U.S. Copyrights
International Copyrights
Uniform Trade Secret Law
Patent Law
Trademark Law
Computer Crime Laws, Federal
Computer Crime Laws, all 50 States
Warranty Disclaimers
Military Computer Acquisition Regs.
Federal Computer Acquisition Regs.
Uniform Commercial Code
Legal Liability of Computer Professionals
Beta Testing Law
BBS Laws
Employer/Employee Relations
Computer and Software Leasing
Credit and Collections
Contract Bidding
Semiconductor (Mask) Work Protection
Arbitration
I.R.S. Issues Relating to Computers
Patenting vs. Copyrighting

Written by a computer-literate attorney

PRIVATE BBS • COMPUTER EMPLOYMENT EXCHANGE • UPDATES

Minimum System Requirements: MS-DOS XT, AT, 386 Compatable, with 512k Ram, Hard Drive

© 1990 Kraft & Byron

Number 34 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The other three methods in **selectableItem** select and deselect objects (**setSelection**), toggle the selected field between TRUE and FALSE (**toggleSelection**), and return the object's string as a function **getMarkedString**.

That function is virtual because I may want to change the format of marked strings later. Until then, the method simply adds '>' to the front of the **strItem**'s string data if the object is selected. To retrieve a plain string, call the inherited **getString**. For a marked string, call **getMarkedString**. Instead of replacing **getString** with a new method that enhances its operation, two separate methods in this case are better than one.

Similar moments occur in non-OOP Pascal. For example, if you write a procedure named **clrscr**, it completely replaces the similarly named procedure in the CRT unit; but if you use a different name like **clearScreen**, you can still call the old routine. There isn't any hard rule about this—you're free to replace existing procedures, functions, and methods by reusing the names. Just be sure that's what you want to do.

The final object for this time is **selectionList**, a descendant of **list**, and one of the more complex objects presented so far:

```
selectionListPtr = ^selectionList;
selectionList = object( list )
   xmin, ymin, xmax, ymax : word;{ Display coords }
   constructor init( x1, y1, x2, y2 : word );
   procedure firstSelectedItem(var noItems:Boolean);
   procedure nextSelectedItem(var pastHead:Boolean);
   procedure setAll( setting : Boolean );
   procedure toggleAll;
   procedure displayItems; virtual;
   procedure selectItems( exitKeys : keySet;
      var lastKey : char ); virtual;
end; { selectionList }
```

A **selectionList** is a list of selectable objects displayed in a fixed location, defined by the four **word** fields: **xmin**, **ymin**, **xmax**, and **ymax**. Constructor **init** initializes the object to the four values passed as arguments. For example, if **myJobs** is a **selectionList** object, a simple statement initializes it to the right half of a typical 80x25 display:

```
myJobs.Init( 40, 1, 80, 25 );
```

Six methods finish the object. **FirstSelectedItem** prepares the list so that the next call to **currentItem**—a method inherited from **list**—returns a pointer to the first **selectedItem**. If **noItems** is true after calling that method, then the list is empty or it contains no selected items. Dealing with extreme cases like these is no different in OOP from other programming paradigms.

Method **nextSelectedItem** advances the list so that **currentItem** returns a pointer to the next **selectedItem**. If true, **pastHead** indicates that previous calls to **nextSelectedItem**

LISTING 3

```
(* uscreen.pas -- (c) 1990 by Tom Swan *)

unit uscreen;
interface
uses crt, dos;

procedure clearWindow( x1, y1, x2, y2 : word );
procedure scrollUp( x1, y1, x2, y2 : word );
procedure scrollDown( x1, y1, x2, y2 : word );

implementation

{ ----- Clear display area to current background color. }
procedure clearWindow( x1, y1, x2, y2 : word );
var
   oldMin, oldMax : word;
begin
   oldMin := crt.windMin;
   oldMax := crt.windMax;
   window( x1, y1, x2, y2 );
   clrscr;
   crt.windMin := oldMin;
   crt.windMax := oldMax
end; { clearWindow }

{ ----- Scroll display area up one line }
procedure scrollUp( x1, y1, x2, y2 : word );
var
   regs : registers;
begin
   with regs do
   begin
      ah := 6;              { BIOS function number }
      al := 1;              { Number of lines to scroll }
      bh := crt.textAttr;   { Attribute for blank line }
      cl := x1 - 1;
      ch := y1 - 1;
      dl := x2 - 1;
      dh := y2 - 1
   end;
   intr( $10, regs )
end; { scrollUp }

{ ----- Scroll display area down one line }
procedure scrollDown( x1, y1, x2, y2 : word );
var
   regs : registers;
begin
   with regs do
   begin
      ah := 7;              { BIOS function number }
      al := 1;              { Number of lines to scroll }
      bh := crt.textAttr;   { Attribute for blank line }
      cl := x1 - 1;
      ch := y1 - 1;
      dl := x2 - 1;
      dh := y2 - 1
   end;
   intr( $10, regs )
end; { scrollDown }

end. { uscreen }
```

LISTING 4

```
(* dirtest.pas -- (c) 1990 by Tom Swan *)

program dirtest;
uses crt, ustritem, udir;
var
   dir : directory;
   lastKey : char;
   name : fileNameStr;
   flag : Boolean;
begin
   clrscr;
   dir.init( '*.*', 10, 3, 40, 18 );
   dir.setAll( FALSE );
   dir.selectItems( [], lastKey );
   gotoxy( 1, 25 );
   writeln( 'Selected files:' );
   dir.firstSelectedItem( flag );
   while not flag {no items or past list head} do
   begin
      name := strItemPtr( dir.currentItem )^.getString;
      write( name, ' ' : 20 - length( name ) );
      dir.nextSelectedItem( flag )
   end;
   writeln;
   dir.done
end. { dirtest }
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

WHICH LANGUAGE DO *YOU* SPEAK??

MXW

Mixed Language Windowing System

1 MXW version is compatible with C, QC, Pascal, Assembler, MS BASIC, QB, and FORTRAN

$59.95

(FREE demo disk available)

mtd
SOFTWARE
development company, ltd.
21 Ralph Ave., N. Brentwood, NY 11717

(516) 434-1979

Number **312** on the Reader Service Card

Build with an Inside! advantage...

Thousands of C, BASIC, FORTRAN and Modula-2 programmers are already using Inside! to improve application performance. You have the best compiler and debugger, now add the performance analysis tool that helps create world class applications!

Inside! analyzes your program in real-time, timing each instance of a function, source line or arbitrary code fragment, all with microsecond accuracy and without source code modification. Utilize the following modes to know where to focus your optimization efforts for maximum reward.

- Function/procedure timing
- Call target timing
- Source Line timing
- Arbitrary event timing
- Non-executed functions or source lines
- MS-DOS function timing
- Overlay and assembly language support

Inside! is available for popular compilers . . .

Call or write today for a free brochure and the *latest list of Inside! products.*

To Place Orders
(800)537-5043
Product Support
(508)478-0499
Visa/Mastercard/C.O.D. Accepted

$125.00 EACH
Satisfaction guaranteed

Paradigm Systems
P.O. Box 152 Milford, Massachusetts 01757
Inside! is a trademark of Paradigm Systems.

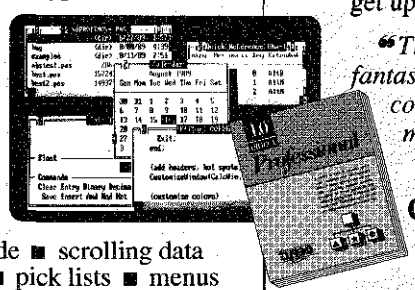
Contact this advertiser directly.

Save "Man-Years of Effort" with Turbo 5.5

Don't Start from Scratch with Object-Oriented Pascal

Object Professional is a huge library of over 200 object types and 2000 methods that will multiply your productivity. Window object types let you use overlapping and resizeable windows. The windows include ■ scrolling data entry screens ■ pick lists ■ menus ■ file selection ■ printed forms ■ help capability and more.

Build your programs using proven data object types like stacks, linked lists, virtual arrays, and more. System-oriented routines provide swappable TSRs in only 6K of RAM, EMS management, and much more.



Object Professional includes clear, comprehensive documentation, on-line help, full source code, technical support, and hot demo programs. Pay NO royalties. You'll get up to speed fast with OOP!

"The range of objects is fantastic. Object Professional could literally save you man-years of effort."
Jeff Duntemann

Object Professional 1.0, only $150.

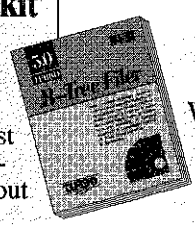
A Multi-User B-Tree Toolkit

Write powerful network compatible databases faster and easier using **B-Tree Filer 5.0**. You'll have the fastest, safest, most flexible databases – no rigid structure, no TSR hassles, no running out of files. And they're compatible with Novell, 3Com, MS-NET, and others.

You get ■ Fixed and variable length records ■ Two billion records per database ■ Up to 100 indexes per index file ■ Fail-safe mode with journaling ■ Units for sorting, browsing, reindexing, and network control.

B-Tree Filer includes full source code, documentation, technical support, and you pay NO royalties.

"B-Tree Filer... a well rounded, feature-rich approach to B-Tree databases."
Computer Language, 1/90



B-Tree Filer 5.0, only $125. (single user)
With network support, $175.

Call toll-free to order.
1-800-333-4160

Satisfaction guaranteed or your money back within 30 days. Add $5 per order for shipping in U.S. and Canada. Inquire about other shipping charges. OPro requires Turbo 5.5. BTF requires Turbo 4.0, 5.0, 5.5, or QuickPascal.

TURBO Power

8AM - 5PM PST Monday through Friday, USA & Canada.
For more information call (408) 438-8608. Fax: (408) 438-8610.
TurboPower Software PO Box 66747 Scotts Valley, CA 95067-0747

Number **25** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

have returned all selected objects. In that case, don't call **currentItem**—it may or may not return a pointer to a selected object.

Together, these two methods make it easy to retrieve all selected objects from a list. If **stock** is a **selectionList** object and **finished** is a Boolean variable, a few lines of code display all selected items in the list:

```
stock.firstSelectedItem( finished );
while not finished do
begin
   writeln(
    strItemPtr( stock.currentItem )^.getString );
   stock.nextSelectedItem( finished )
end;
```

It's necessary to cast **stock.currentItem** in the **writeln** statement to type **strItemPtr** to inform the compiler that **getString** belongs to the object at this address. As I've mentioned before, such casts are unfortunately necessary in OOP when calling inherited methods that return pointers of one type that actually address object descendants of another.

Two other methods in **selectionList** change and toggle an object's selection status. Call **setAll(t)** with **t=TRUE** to select or **t=FALSE** to deselect all objects. Call **toggleAll** to toggle all objects on and off. The method implementations show the correct way to scan an entire list of **selectableItem** objects.

The final two **selectionList** methods perform most input and output actions. **DisplayItems** displays a list of objects within the boundaries passed to the list's constructor. Be aware that this method affects the next call to **currentItem**. If you need to preserve the current list position, save and restore the inherited **cip** field in a variable of type **pointer**:

```
oldCip := stock.cip;
stock.displayItems;
stock.cip := oldCip;
```

The final method, **selectItems**, appears to break my earlier 25-line rule, but it gets me off the hook by using five nested procedures to divide the code into manageable chunks. Call **selectItems** to select items from a list of **selectableItem** objects. The method scrolls the list as necessary, and lets you press <Space> or <Enter> to toggle selected items on and off. Hit <Esc> when done. You can then use the technique outlined before to retrieve selected objects.

Pass two arguments to **selectItems**: a set of characters to end selection processing and a character variable—I'll call it **lastkey**, although any name will do:

```
stock.selectItems( ['C','Q'], lastkey );
```

The character set might contain commands to process—a feature that will be more important later. When **selectItems** returns, **lastkey** equals the character that ended the method call. Because you can always press <Esc> to end **selectItems**, **lastKey** will equal **chr(27)** if the character-set argument is empty ([]).

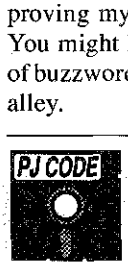
USELECT.PAS needs a few non-OOP procedures to do its thing—that is, to clear and scroll a window up and down—handled by Listing 3, USCREEN.PAS. Since my goal for this project is to get in deep with OOP, I won't explain USCREEN.PAS here. The code is straightforward, and the comments in the listing explain what the procedures do.

SAMPLING THE WARES

Listing 4, DIRTEST.PAS, shows how to use some of the methods in USELECT and UDIR. The program lists files in the current directory and lets you select one or more filenames by pressing <Space> or <Enter>. When you press <Esc>, a **while** loop displays the names you selected.

Even though you may have written your own routines to display directories, if your non-OOP Pascal closet is like mine, it holds a lot of code that's suitable for copying into new programs—but not without making a few alterations.

OOP can change that. My new objects reuse their ancestors. And, by making objects usable from the start—and also by applying a little self-criticism on overgrown code—I'm improving my ability to discover new reusable object classes. You might have gathered from this article that I'm not fond of buzzwords. But I'll keep "reusability."—it's right up OOP's alley.



All code in this issue is available through our bulletin board services or directly from *PJ* on disk. For more information see pages 8 and 88.

* * NEW! * *

Complete Modem Communications
for TURBO PASCAL Programmers

Introducing
Sugar Coated Serial™

A complete library of Turbo PASCAL procedures, functions and low-level assembly routines that allow you to EASILY incorporate MODEM communications into your application.

Features:

- All source code included (.PAS & .ASM).
- No knowledge of assembler required.
- No royalties or fees for use in your programs.
- Easily do XMODEM, YMODEM, & Telink file transfers (including multiple files with YMODEM.)
- Pull-down menu communications program included that uses only routines from this library.
- Complete user manual including many examples.

For IBM PC/XT/AT compatibles using T.P v5.0 or >

* Complete System: $109.95
* Demo Disk: $ 5.95
* Information Pak: Free

Shipping included, IL residents add 6.25% sales tax. Specify disk size.

M/C Visa

Send Check, M.O. or Credit Card info with order to:

Goin Places Software
Suite 145
1202 N West 75th Street
Downers Grove IL 60516

For phone orders or info call: (708) 759-1270

SUBSCRIBE NOW

Number 18 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SOFTWARE DEVELOPERS: ARE YOUR PRODUCTS TOO HARD TO INSTALL?

In today's competitive market, your product **must** be easy to use -- immediately. It is no longer commercially acceptable to tell the end-user to use batch files, COPY, DISKCOPY, RESTORE, or to require wading through paragraphs of text in the manual. The installation process is something that a program developer **should** and now **can easily** completely automate.

INSTALL 2.3 is a bullet-proof automatic installation program you can distribute - royalty free - with your product • Features: Up to 4.3 gigabyte file sizes • Windowing/color user interface • **Full C source** with project, make, link, and batch files • File integrity verification • Intelligent handling of virtually all errors (open disk drive doors, full disks, unformatted disks, etc) • Hard & floppy disk installation • Multiple source & target disks • Multiple target subdirectories (which *INSTALL* will create if necessary) • Unlimited free tech support • Granular installation • Easy method to add-in new functionality • Ability to execute DOS commands • Can intelligently modify CONFIG.SYS & AUTOEXEC.BAT files • Includes **data compressing file librarian** • **NEW** Automatically detects and can base the installation on the CPU (86/286/386), DOS version, display (CGA/B&W/MONO/EGA/VGA), NetBIOS, ANSI.SYS, LIM, etc.

INSTALL offers 3 years of proven reliability that has been used to install over 500,000 copies of products including some of the most prestigious in the industry. We would like to add your name to that list.

ORDER NOW TOLL FREE 1-800-331-2783

KNOWLEDGE DYNAMICS CORPORATION
Highway Contract 4 Box 185-H, Canyon Lake, TX 78133-3508
Voice: 1-512-964-3994 FAX: 1-512-964-3958
MasterCard/VISA/COD/Purchase Order **$149.95**

Number **295** on the Reader Service Card

PRINTER GRAPHICS IN HIGH RESOLUTION!

GRAPHLINK™ Printer Graphics Toolkit

GraphLink™ lets your programs tap the *full potential* of printers to produce dazzling *printed* graphics in *high-resolution!* Compare the features ...

- Uses the PRINTER'S resolution, not the SCREEN'S!
- Can produce 300 dpi images on laser printers
- A linkable library of 80+ routines for building and printing images
- Provides line, bar, arc, pie-slice, pattern, and bitblt routines, fully-scalable fonts and much more!
- Works with any screen graphics software
- Emulates the Borland Graphics Interface (great for BGI programs!)
- Can *automatically* build the printed image, by intercepting your screen graphics calls (Turbo Pascal version)
- Supports the most popular laser and dot matrix printers
- Supports Turbo Pascal 4.0 and higher. Available soon for Turbo C, Microsoft C, and Quick C
- Only ***$125*** for the single-user version, or ***$250*** for GraphLink *Professional*, which includes an unlimited distribution license. (Shipping & Handling: USA $5, Canada $10, Other $15.)

30 Day No Risk Money-Back Guarantee

GRAPHLINK
The Printer Graphics Solution™

VISITECH SOFTWARE
D5 3807 Ridgewood Court
Pittsburgh, PA 15239 USA
(412) 733-4775

Number **304** on the Reader Service Card

TechnoJock's Turbo Toolkit

Attention Turbo Pascal programmers.....

Cut your program development time in half and create sophisticated user interfaces in minutes with *TechnoJock's Turbo Toolkit!*

The *Toolkit* contains twelve units with all source code included, and a 300 page manual full of examples. These range from a comprehensive unit that provides a Sidekick Plus style directory window with all the expected sorting features, to a basic procedure that writes text centered on the screen.

And it's easy to use! If you've bought other toolkits but didn't use them because of all their complicated rules ... TTT is for you.

TECHNOJOCK SOFTWARE

- **ultra fast screen writing**
- **EGA/VGA extended lines**
- **exploding boxes**
- **virtual screens**
- **sorted directory listings**
- **comprehensive form input**
- **pulldown menus**
- **nested menus**
- **string manipulation**
- **windows**
- **full mouse support**
- **keyboard control**
- **date manipulation**

"TechnoJock's Turbo Toolkit may just cut your work in half! It provides easy-to-implement procedures.... designed for all Turbo Pascal programmers. With this Toolkit everyone becomes a TechnoJock."
Turbo Users Group

"Here, despite the whimsical name, is shareware at its best: an innovative product, a superb manual, quick mail-order service, and a reasonable price... If you're programming in Turbo Pascal, you absolutely must have TTT!"
Public-Domain Software & Shareware 2nd edition

"Looks great! We have decided to implement your Toolkit in in-house programs for Tech Support."
Rich Kubota
Technical Support
Borland International

TechnoJock Software, Inc.
PO Box 820927
Houston, TX 77282-0927

Orders: 1-800-242-4775
Enquiries: (713) 493-6354

Only $49.95!

Plus $5.00 shipping and handling (Canada $10.00, overseas $15.00). Texas residents add 8% Sales Tax.

Send checks to TechnoJock Software. Send Visa/Mastercard details to: Public (Software) Library PO Box 35705 Houston TX 77232-5705

Number **289** on the Reader Service Card

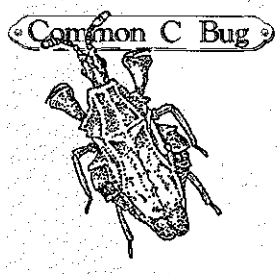
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

T-SHIRTS

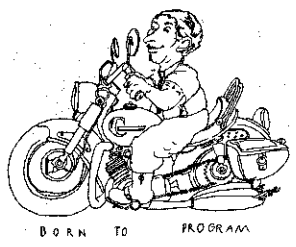
Multi-color imprints on the highest quality 100% cotton T-shirts.

SIZES: S, M, L, XL, XXL

$12.00 each – includes shipping & handling

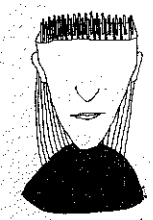


COMMON C BUG



BORN TO PROGRAM

PARALLEL PORTS



RS-232 HEAD

BUGS RULE 4 color imprint on a white shirt

PJ ISSUE SPECIAL!

CODE **$8.00**
DISKS
REG $10.00 ea.

PJ Source Code on Disk . . .

All the code that fits in the magazine, we print—but there's even more on each issue disk! Get any source code disk from issue 5.5 (September/October 1987) on or Michael Abrash's EGA or VGA code for just $8.00 each. (See page 8 for information on a *PJ* Code DiskScription—6 issue disks for $39.95, or just $6.66 each.) Issue-special prices good through 6-30-90.

BACK ISSUES

$7.00 each PREPAID ONLY

A few of the Back Issues currently available

Vol. 6.6: PC DOS 4.0; OS/2 vs. UNIX; VGA screen saving and color selection; MASM quirks; addressing extra memory for DOS; DOS device drivers debugging and information utility; proposed ANSI C language standard; Ray Duncan on Forth; Ada data types and exceptions; PC floating-point hardware and software.

Vol. 7.1: **GRAPHICS:** High-resolution VGA color modes; graphics processors; modifying type constants stored in Turbo Pascal 4.0 and 5.0 compiled EXE code; comparing Pascal overlay managers; JPI's TopSpeed Modula-2; faster QuickBASIC code; Microsoft's CodeView debugger; Lattice C V3.3; smart terminal emulations in Windows; the COBOL-85 ANSI standard; Pink Noise—industry news and reviews.

Vol. 7.2: **OPERATING SYSTEMS:** undocumented DOS features; fast 80386 block moves under DOS; OS/2 dynamic link libraries; OS/2 multitasking; debugging utility for PM and Windows; EGA and VGA animation; disk formatting in Pascal; how to become a reseller; reviews of windowing and graphics tools, disassemblers, and *The 80386 Book*.

Vol. 7.4: **COMMUNICATIONS:** "Pink Noise," by Frank Greco; Michael Abrash's "On Graphics"; a simple terminal program in PM; communications and protocols; better coding conventions for assembly language; implementing the Boyer–Moore search algorithm in assembly language; database normalization theory and practice; Turbo Pascal 5.5 development tools; Lint utilities for the PC; a millisecond timer for BASIC; exit codes for QuickBASIC; reviews of RamNet, DOS/16M, PCX *Technical Reference Manual* and Programmer's Toolkit, GRASP 3.1, AXOS, and *Microsoft Mouse Programmer's Reference*.

Vol. 7.5: **OBJECT-ORIENTED PROGRAMMING:** Class libraries in C++; from C to objects; improved C++ code generation; Hal Hardenbergh on the RISC/CISC schism; handling version management on large projects; "Still-Higher 256-Color VGA Resolutions," by Michael Abrash; a library system for assembly language; adding world scaling to Turbo Pascal's BGI graphics primitives; numeric extensions to C; reviews of 386 development environments and tools, mini-case tools, and Robert Sedgewick's book *Algorithms*.

Vol. 7.6: **USER INTERFACES:** Windows terminal program; the CURSES C library; using two video displays with Turbo Pascal; Michael Abrash on VGA and EGA split screens; memory models and static variables for assembly language; ANSI C standard; Aztec C version 4.2; using object-oriented programming to create a file viewer; accessing large memory spaces from DOS on the 80386; making use of QuickBASIC's string-searching algorithm; reviews of user-interface tools and mini-Case tools; the *System BIOS for IBM PC/XT/AT*, series, by Phoenix Technologies.

Vol. 8.1: **NETWORKING:** Frank Greco on the Network File System; an overview of NetBIOS on the PC, excerpted from *C Programmer's Guide to NetBIOS*, by W. David Schwaderer; fast circle drawing on the VGA, by Michael Abrash; "Dynamic Data Exchange in Windows," by William S. Hall; an assembly language program for effective use of dynamic memory, by John Otken; taking advantage of DESQview API features; computing time in C; Tom Swan on creating a menu system with Turbo Pascal 5.5; government agencies as consulting customers.

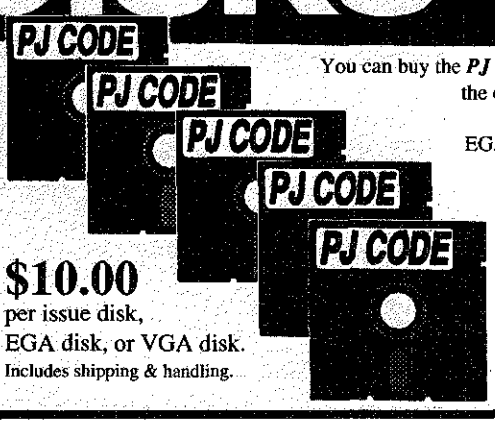
Vol. 8.2: **GRAPHICS:** Michael Abrash's "Faster Circles for the VGA"; the TIGA interface from TI; understanding the EGA palette; programming the 800x600x16 Super-VGA mode; dealing with defaults; dynamic data exchange for communications; John Otken's "Manipulating Large Data"; fast file I/O in QuickBASIC; industry news; 1989 article index.

10% discount on any order of 10 or more T-Shirts, Disks or Back Issues.

All mail orders must be accompanied by check or Money Order -- PREPAID ONLY.

CODE DISKS

Use the order form on the card opposite.
Order by phone with your VISA/MC.
CALL TOLL FREE
800/234-0386 ext 53



$10.00
per issue disk,
EGA disk, or VGA disk.
Includes shipping & handling.

You can buy the *PJ* code listings from volume 5.5 to the current issue. Code disks for the Michael Abrash "On Graphics" EGA/VGA series are also available.

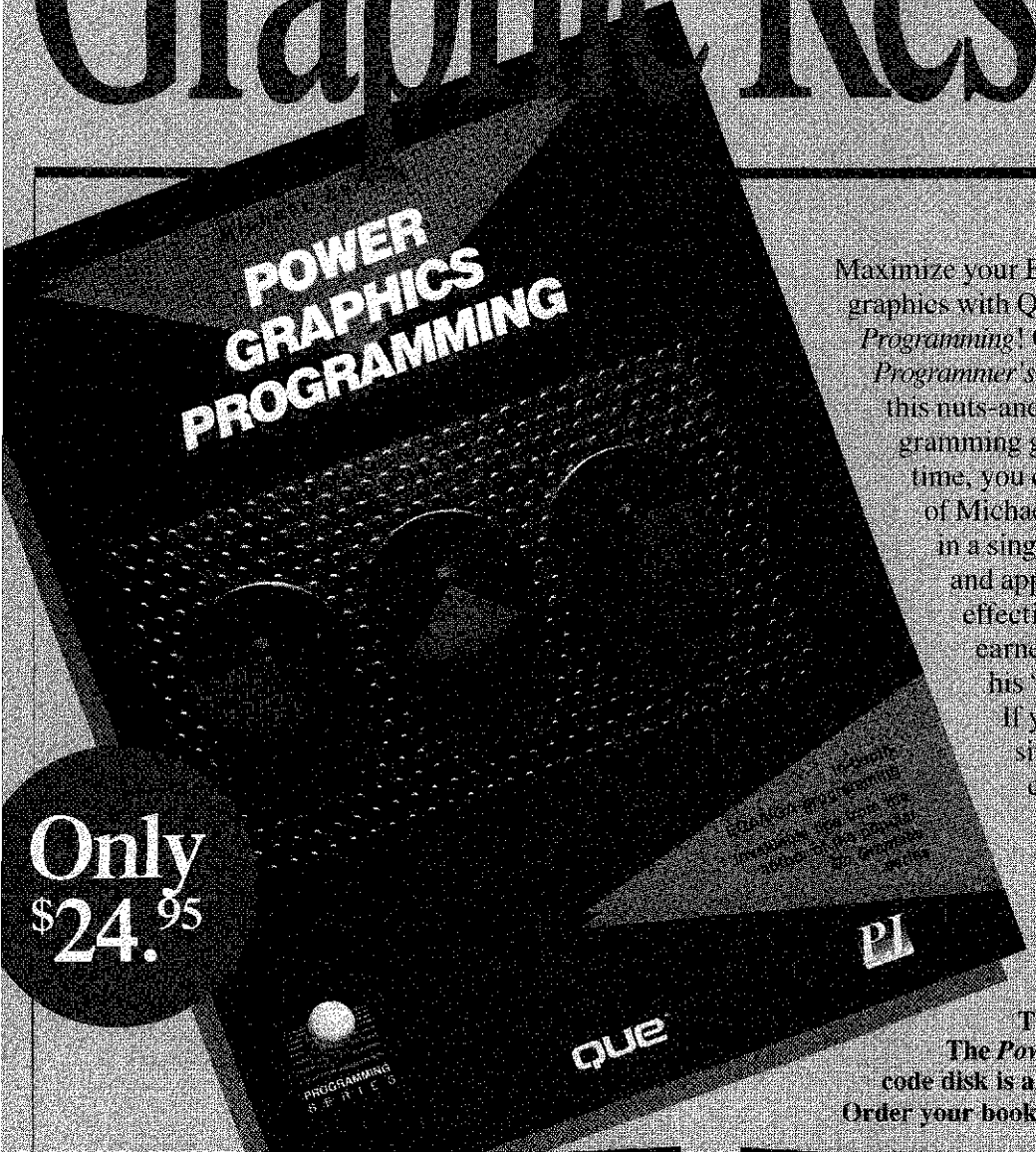
Diskettes are available for the following issues:

5.5	6.3	7.1	7.5	8.3
5.6	6.4	7.2	7.6	EGA
6.1	6.5	7.3	8.1	VGA
6.2	6.6	7.4	8.2	

360K MS-DOS disks standard; other formats available.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Graphic Results!

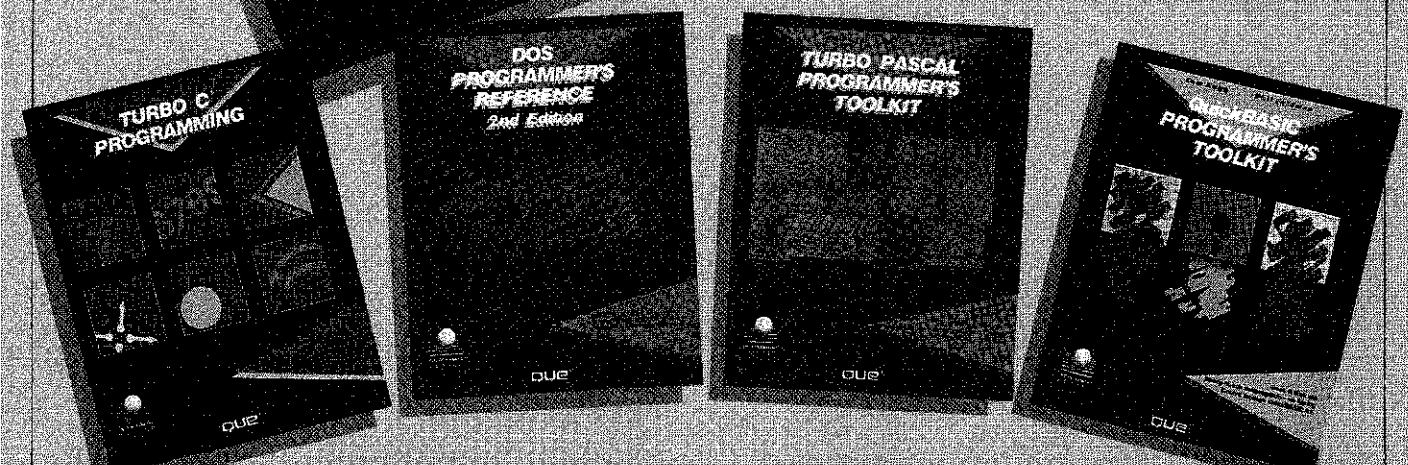


Only $24.95

Maximize your EGA/VGA programming graphics with Que's *Power Graphics Programming*! Que has teamed up with *Programmer's Journal* to bring you this nuts-and-bolts graphics programming guide. Now, for the first time, you can have the expertise of Michael Abrash consolidated in a single volume. His insights and approach to efficient, effective programming have earned him high acclaim for his "On Graphics" series. If you are seeking that elusive graphic edge, you can't be without *Power Graphics Programming*!

Order yours directly from *Programmer's Journal*

To order call 800-234-0386. The *Power Graphics Programming* code disk is also available for $14.95. Order your book and code disk today!



Look for these other practical Programming books from Que

Available in better bookstores and computer stores everywhere. Or call 800-428-5331, ext. PJ01.

que®

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ PRESSROOM

SPONTANEOUS ASSEMBLY

BaseTwo Development has released Spontaneous Assembly, a library for serious assembly language development. The library contains more than 600 assembly language functions for the MS-DOS 80X86 environment. Library functions provide text windowing, critical-error handling, string, memory, and array manipulation, date and time functions, a true millisecond clock, large-integer math using quad words, version-independent DOS functions, and full file and directory control. Features include register-based parameter passing, integration with high-level languages (for example, Microsoft and Borland), and support for all of the standard memory models. Source code and a complete manual are provided with the package, which has a list price of $199. BaseTwo Development offers a 60-day money-back guarantee.

BaseTwo Development
11 East 200 North
Orem, Utah 84057
801-222-9500

TURBO DEBUGGER & TOOLS

Borland International has unveiled a new set of programming tools designed to improve the way developers build and test complex software products. Borland'shigh-performance Turbo Debugger & Tools, includes an improved Turbo Debugger 2.0, which supports object-oriented debugging, an updated assembler (Turbo Assembler 2.0), and the new interactive Turbo Profiler. This tool set is designed to give professional developers a major advancement in the art of finding and fixing software bugs, analyzing run-time execution, and optimizing software code.

The new Turbo Debugger supports most popular DOS compilers, including products from Borland, Microsoft, and WATCOM. It features reverse execution, program record and playback, an easy-to-use interface with mouse support, object-oriented programming support, 80286 and 80386 protected-mode operation, debugging support for terminate-and-stay-resident (TSR) programs, and device drivers.

The new 80286 protected mode uses the OS/286 DOS Extender from Eclipse (formerly A. I. Architects) and uses only 72K of main memory. Turbo Profiler is interactive and features both active analysis (line and routine timing) and passive profiling (sampling), multiple views, an overlapping windows interface with mouse support, and profiling reports. Turbo Assembler features speeds up to twice as fast as MASM, 100-percent compatibility with MASM, multipass assemblies with NOP squishing, 80486 support, and easy high-level language interface. Turbo Debugger & Tools is scheduled to be released during the second quarter, with a suggested retail price of $149.95.

Borland International Inc.
1800 Green Hills Road
Scotts Valley, CA 95066
408-439-9273

FROM TURBO TO POWERBASIC

Borland has transferred Turbo Basic's future-development and new-version-publishing rights to their BASIC compiler back to its original developer, Robert Zale. Zale in turn, has reached agreement with **SPECTRA Publishing** to publish a new, much-updated version of Turbo Basic under the name PowerBasic. SPECTRA Publishing has released PowerBasic 2.0, a substantial upgrade of the former Borland product.

PowerBasic now supports new data types, including BCD, fixed and floating point, a new extended-precision (80-bit) floating point, quad (64-bit) words, and "flex" string variables. Other significant new features include dynamic data structures, string handling for strings greater than 64K, built-in array manipulation (sort, scan, insert, and delete features), modular compilation, and linking.

PowerBasic provides a comprehensive programming environment with an integrated editor, debugger, compiler, and linker, as well as command line versions of these programs. SPECTRA offers full technical support. The upgraded version of PowerBasic is priced at $109.95. Registered users of Turbo Basic 1.1 may upgrade to PowerBasic for $50. SPECTRA Publishing is a newly-created division of PC-SIG Inc. (founded in 1983), the largest U.S. publisher of user-supported software.

SPECTRA Publishing
1030D East Duane Ave.
Sunnyvale, CA 94086
408-730-9291

MASTERFUL DISASSEMBLER

C. C. Software has released version 2.3 of Masterful Disassembler, their interactive disassembler for the 80X86. Version 2.3 represents a major step forward in the art of program disassembly. In addition to increasing the intelligence of the disassembler, version 2.3 adds MACRO capabilities. Disassembly macros are the counterpart of assembler macros. Sequences of instructions are combined to form macro instructions with a flexible argument arrangement. Macros greatly simplify the disassembly process, particularly for code generated from high-level languages, making the code easier to understand. A separate macro compiler is provided so that users can create their own macro definition libraries from English-like source files. All aspects of the disassembly process and the use of macros is explained in two indexed manuals. MD86 version 2.3 is $67.50.

C. C. Software
1907 Alvarado Ave.
Walnut Creek, CA 94596
415-939-8153

META3D FOR 3D GRAPHICS

F. G. Soft has released META3D, a complete 3-D graphics library for use with the MetaWINDOW library from MetaGraphics. META3D provides powerful 3-D graphics capabilities. Programs written in C or FORTRAN can call META3D routines for plotting 3-D views and other 2-D/3-D manipulations along with other MetaWINDOW routines for graphics manipulation. The META3D and the stand-alone library, CGL3D, together cost $199.

F. G. Soft
P.O. Box 2152
Hawthorne, CA 90250-2152
213-542-5461

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A SPELLING CHECKER

Geller Software Laboratories has released SpellCode—the first spelling checker specifically for programmers. SpellCode is designed to check and correct spelling in text strings, help messages, and other text appearing in program code, databases, and spreadsheets. SpellCode works with dBASE files, C, Pascal, BASIC, and many other languages and recognizes their keywords. SpellCode's understanding of the programming languages allows it to check only the text displayed to users or, alternatively, the entire contents of program files. It can check and correct the contents of character and memo fields in DBF database files as well as in Lotus and Symphony worksheet cells. The program features a flexible dictionary system. General-purpose checking modes work with Ada, COBOL, PL/I, and FORTRAN. SpellCode supports an easy-to-use windowing interface.

Jake Geller, the principal developer, is already well-known for an excellent series of dBASE tools, QuickCode, from Fox and Geller. SpellCode has a list price of $99.95.

Geller Software Laboratories Inc.
35 Stephen Street
Montclair, NJ 07042
201-746-8467

GREENLEAF COMMLIB 3.0

Greenleaf Software has released a new version of its popular communications library. This library of over 150 C functions furnishes programmers with significant new capacities in communication applications. CommLib 3.0 includes Kermit, XModem, X-Modem 1K, and YModem batch file transfer protocols. The library has full support for automatic RTS-CTS hardware flow control (required for 9600-baud modems), Hayes modem control functions, and XON/XOFF software flow control.

The library of functions is highly granular, for minimal overhead. CommLib also features advanced capabilities such as the ability to filter up to three codes from the receive stream, store status along with data in a special "Widetrack Receive" mode, and ignore or respond to modem status at the interrupt service level. As with other Greenleaf libraries, no royalties are charged, and all source code is included. CommLib is $299.

Greenleaf Software, Inc.
16479 Dallas Parkway, Suite 570
Dallas TX 75248
800-523-9830

VM/386 MULTIUSER

IGC has released VM/386 MultiUser Starter—a software program that allows a user to add two text-based terminals to a 386 PC to obtain a three-user plug-and-play multiuser solution that is completely DOS compatible. To install VM/386 MultiUser Starter, you simply connect two monochrome, text-based terminals into the COM ports of the 386 PC and install the software, and the system is up and running. Users start their own copies of DOS with autoexec and config files, and each user may have terminate-and-stay-resident (TSR) programs. And each user may reboot (with <Ctrl-Alt-Del>) his or her own workstation without affecting other users. VM/386 Multiuser Starter provides a simple, low-cost alternative to small LAN systems. Customers who need to add more users to the system later or who want graphics capabilities can upgrade to VM/386 Multiuser, which supports 32 users of text stations, graphics stations, or a combination. VM/386 MultiUser Starter sells for $395, while the full VM/386 Multi-User is $895.

IGC
4800 Great America Parkway
Santa Clara, CA 94054-1221
408-986-8373

FORTRAN 8X DRAFT

As a service, **Lahey Computer Systems** is selling the latest version of the FORTRAN 8X Draft Standard submitted by the ANSI X3J3 Committee at a reduced price of $23. This is considerably less than the $60-plus price from Global Engineering, the standard source for ANSI draft documents. Lahey is also selling proceedings from the FORTRAN 8X Implementation Workshop meetings held in September 1989, for $23. Both documents may be purchased together for $40. Lahey has been working with MetaWare to merge the Lahey FORTRAN compiler front-ends with MetaWare code generators on various new workstations and CPUs.

Lahey Computer Systems Inc.
P.O. Box 6091
Incline Village, NV 89450
702-831-2500

NEW LATTICE C COMPILERS

Lattice has released its new 80286 C Development System for DOS & OS/2 and has announced an April release of the new 80386 C Development System for DOS & OS/2. The new 80286 C development system features a complete programming environment including a DOS extender, compiler, assembler, debugger, editor, global optimizer, programming utilities, and nearly 800 library functions. The compiler and all the utilities in the new package have been performance-tuned for 80286 and 80386 systems. The Lattice 80286 C Development System runs under DOS, Extended DOS, or OS/2 to create a single executable program that can run under DOS, Extended DOS, or OS/2. The package includes a royalty-free DOS extender, which developers may include with their software at no charge. Lattice's DOS Extender allows user to run programs of up to nearly 16 megabytes. A configuration optimizer is also included to assure maximum performance of the DOS extender.

The Lattice 80386 C Development System will take full advantage of the 80386 and 80486 32-bit processing for optimum performance of the compiler and programs generated. It runs under Lattice's 80386 Extended DOS, Phar Lap's DOSIExtender, or OS/2 Version 2 and generates code that can run under Extended DOS or OS/2 Version 2. The compiler is upwardly compatible and so will accept source code written for DOS or OS/2 Version 1 and optimize it to take advantage of an 80386- or 80486-based system's power. With these new compilers, Lattice is introducing its unique Extended Family Mode, in which a single program can run under either Extended DOS or

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

OS/2. Software developers need to maintain only one program, even though their customers may use the program under several different operating systems. The 80286 C Development System is priced at $495; the 80386 C Development System is $900.

Lattice, Inc.
2500 South Highland Avenue
Lombard, IL 60418
708-916-1600

MICROSOFT OS/2 PM TOOLKIT

Microsoft has released a comprehensive set of state-of-the-art programming tools for OS/2 Presentation Manager version 1.2 to complement its OS/2 language compiler family and operating system. The MS OS/2 PM Toolkit version 1.2 combines a collection of enhanced graphical tools, including over three megabytes of on-line code, with a library of on-line and hard-copy tutorials and references. The toolkit updates PM tools to implement advanced features of OS/2 1.2, including the High-Performance File System (HPFS). Tools include an icon editor (which supports 256 colors and device independence), a dialog-box editor, a font editor, a resource compiler, and an on-line sample-code toolbox. The kit also comes with a complete set of printed and on-line references and tutorials as well as a recommended CUA/SAA (Common User Access/System Application Architecture) printed style guide. The OS/2 PM Toolkit is priced at $500.

Microsoft Corporation
16011 NE 36th Way
Box 97017
Redmond, WA 98073-9717
800-421-8006

QUANTASM POWER LIB

Quantasm Corporation recently announced a new product, called QuantasmPower Lib. This function library for assembly language programmers (from novice to expert) contains more than 250 assembly language routines written for speed and compactness. Some of the functions include a character-based menu and windowing system, extended precision math, more than 76 string-handling functions, date and time functions, encryption and decryption routines, and character drawing. High-level language interface routines are also provided. Quantasm Power Lib is so compact (according to Quantasm Corporation) that a fully functional program with nested windows, drop shadows, and menus can be written in less then 5K of code. The library is priced at $99.95; it is available with source code included for $299.95.

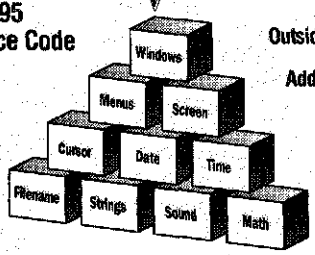
Quantasm Corporation
19855 Stevens Cr. Blvd., Suite 154
Cupertino, CA 95014
408-244-6826

ATTENTION ASSEMBLY LANGUAGE PROGRAMMERS

BUILDING BLOCKS FOR THE ASSEMBLY LANGUAGE PROGRAMMER

QUANTASM POWER LIB

Price $99.95
With Source Code
$299.95



S/H $5.00
Outside U.S. $15.00
CA Residence
Add 7% Sales Tax
MC and VISA

- *Over 256 Assembly Language Routines*
- *No Need to Re-Invent the Wheel Every Time You Write a New Application*
- *Permits Faster Application Development*
- *Overlapping Windowing System in Less Than 3K*
- *Fast and Compact*
- *Ideal For Programming TSR's*
- *C and Pascal Interfaces Available*
- *MASM 5.X Compatible*

QUANTASM CORPORATION
19855 Srevens Creek Blvd, Suite 154
Cupertino, CA 95014
(408) 244-6826

ALSO AVAILABLE:
ASMFLOW $99.95
An Assembly Language Flow Charting and Source Code Analysis Tool

Number 252 on the Reader Service Card

Total control with LMI FORTH™

For Programming Professionals: an expanding family of compatible, high-performance compilers for microcomputers

For Development:
Interactive Forth-83 Interpreter/Compilers for MS-DOS, OS/2, and the 80386
- 16-bit and 32-bit implementations
- Full screen editor and assembler
- Uses standard operating system files
- 500 page manual written in plain English
- Support for graphics, floating point, native code generation

For Applications: Forth-83 Metacompiler
- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 8086, 68000, 6502, 8051, 8096, 1802, 6303, 6809, 68HC11,34010, V25, RTX-2000
- No license fee or royalty for compiled applications

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone Credit Card Orders to: (213) 306-7412
FAX: (213) 301-0761

Number 63 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ BILLBOARD

FIGHT PIRACY & PROTECT YOUR PROGRAM $$$'s!

Since 1986, companies worldwide have been choosing Az-Tech security products. If you demand the strongest protection available, why not choose one of these "proven leaders":

- **EVERLOCK** Copy Protection
- **EVERTRAK** Software Security
- **EVERKEY** Hardware "Key" Software Security

For IBM and Compatibles.
30 day money back guarantee
Free info and demo disk available.

Az-Tech Software, Inc.
305 East Franklin
Richmond, MO 64085
(800) 227-0644 (816) 776-2700
FAX (816) 776-8398

Number **51** on the Reader Service Card

HUGE Professional Shareware Libraries

Comprehensive, indexed collections of all known Public Domain and Shareware utilities for PC professionals, updated quarterly with the latest products. Includes extensive disk based or printed directories with multiple search options. Each library designed specifically with the PC professional in mind. Computer Language called our dBase product "*possibly the most productive $100 you'll ever spend*". VISA, MC, POs, COD.

Product	Disks/Files	Price
dBase/dBase Compilers	56+/1100+	$99.50
AutoCAD	18+/560+	$99.50
C	19+/400+	$49.50
Turbo Pascal	19+/200+	$49.50
Ventura DTP	21+/170+	$49.50
Lotus 123	17+/200+	$49.50
Assembler	10+/170+	$49.50
Games	31+/200+	$49.50
Novell Netware		$49.50
WordPerfect		$49.50
Utilities for DOS Gurus		$49.50

EMS, 4505 Buckhurst Ct.
Olney, MD 20832 USA
(301) 924-3594 Fax: (301) 963-2708
France (1)47763872, Germany 0911/338844

Number **353** on the Reader Service Card

EZ-COPY PLUS

Diskette Duplicator

•2X+ faster than DOS •Read diskette once, then... •Quickly & accurately mass duplicate 5.25" & 3.5" disks (360K, 720K, 1.2M, 1.44M) on your own PC/XT/AT/etc. © • Supports up to 4 drives with automatic start and switching. •Formats, copies, verifies, etc. • Partial copies too. •Even faster with pre-formatted disks. • Serializes disks and/or labels, in 1 smooth operation. • Save disk images to HD, more... • Replaces dedicated hardware worth $1000s. **Only $129** +s/h

V/MC/AX Orders & Brochures Toll Free:
1•800•873•2799 xPJ590
INFO 1-713-280-9900

S&H: $6 USA-48; $10 N.Amer; $20 Foreign.
EZX Publ., 917 Oakgrove Dr., Ste PJ590, Houston, TX 77058 •FAX: 1-713-280-0525

Number **352** on the Reader Service Card

FREE CATALOG!

LATEST AWARD BIOS
PC/XT, 286, 386

COMPILED BASIC TOOLS
Libraries, Utilities
for **QuickBASIC 4.5**

HARDWARE UPGRADES
for PCs & PC/XTs

(800) 423-3400 - (412) 782-0384
KOMPUTERWERK INC.
851 Parkview Blvd.
Pittsburgh, PA 15215

Number **155** on the Reader Service Card

RELATIONAL DATABASE LIBRARY

For TurboC, Microsoft C on DOS systems. The 114 functions in **"c_wndw"** and **"c_ndx"** libraries support:

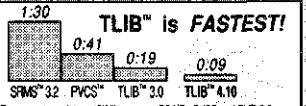
- Multiple B-tree index files
- Stacked, overlapping windows
- dBase .DBF and .NDX file formats
- Multi-level file relationships
- Pull-down and pop-up menus
- Hashed random file access
- Data dictionary field access

Libraries $99.50; Source $199.50; VISA/MC

Marietta Systems, Inc.
P.O. Box 71506
Marietta, GA 30007

(404) 565-1560

Number **223** on the Reader Service Card



Times are to update a 45K library on a PC/XT. PVCS and TLIB 3.0 are from Sept 87 *PC Tech Journal*. SRMS and TLIB 4.10 are later versions.

TLIB™ is *BEST!*

"Do not be fooled by the fact that this is the least expensive of the five packages reviewed here - TLIB has features and power to spare" John Rex, **Computer Language**

"TLIB is a great system" J. Vallino, **PC Tech J**

Full-Featured Version Control for Software Professionals. Check-in/out locking. Branching. Keywords. Wildcard and list-of-file support. Can merge parallel changes and undo intermediate revisions. Network and WORM support. **Mainframe compatible** deltas for Pansophic, ADR, IBM, Sperry. Integrated Landon Dyer PD **MAKE**.

MS-DOS 2.x, 3.x, 4.x **Just $99.95** + $5 s/h Visa/MC
5 station LAN license $299.95 + $5 s/h, call for other sizes

BURTON SYSTEMS SOFTWARE
PO Box 4156, Cary, NC 27519 (919) 856-0475

Number **288** on the Reader Service Card

LAI

Logical Alternatives Incorporated

SMP's amazing windows, now available in *Basic*!

FREE DEMO

(814) 234-8088
BBS (814) 234-1881

Contact this advertiser directly.

FREE TRIAL

Pinnacle File Manager

We guarantee immediate productivity gains from using our C file manager library. Simple table-and-column view of data with advanced B-Tree indexing and hi-speed cache frees your brain to solve application-specific problems.

Vermont Database Corp.
RR3 Box 6516
Stowe, VT 05672

802-253-4437

Number **328** on the Reader Service Card

1000'S OF NEW COMPUTER BOOKS UP TO 40% OFF

New titles for the professionals including books for PCs and compatibles, MAC, UNIX, minis, and mainframes. DOS, OS/2, X Window, X Open, and UNIX V/386. Ada, C, C++, Modula-2, Pascal, LISP, FORTH, ASM, and other languages.

Desktop publishing, word processing, CAD, graphics, image processing, and hardware.

Send five 25¢ stamps for complete 70-page catalogue.

Computer Books
P.O. Box 70746
Sunnyvale, CA 94086-0746

RS# 340

---

dBASE FILE ACCESS FROM C

Code Base 4 is a library of C routines which gives complete dBASE and Clipper functionality and file compatibility. The routines correspond directly to dBASE commands. Create multi-user applications using pulldown menus, popup windows and data entry.

Code Base 4 supports DOS, SCO Xenix, OS/2 and Microsoft Windows. $295 with Source! FREE DEMO.

Sequiter Software Inc.
P.O. Box 5659, Station L
Edmonton, Alberta
Canada T6C 4G1
403-439-8171
Fax: 403-433-7460

RS# 269

CSOURCE APPLICATION GENERATOR

C code WYSIWYG screen, report, & program generator. Library code may be enhanced for custom generation. Supports multi-user development & DOS 3.1/Novell networks. Produces small-sized DATA ENTRY, REPORT, MENU, INDEX REBUILD & FILE RESTRUCTURE programs with multiple screens & files. Very fast development & execution speeds. Requires 256K RAM, DOS 2.1 or later, Faircom Ctree or Novell Btrieve. For Turbo or Microsoft C 5.0. No royalties. ALL generated applications source is supplied.

$395.00 executable, $595.00 full source code. Multi-programmer licenses.

MBAC
13320 Castana Ave.
Downey, CA 90242
213-531-7259

RS# 291

WHITNEY EDITOR

Small and fast. Supports EMS. Split-screen editing. Regular expression search and tagged replace. Shift-block mode. Condensed-display mode. One-key compile. Mouse support. Features for writing documentation. Run on DOS programs with swap. Supports EGA 43 and VGA 12, 14, 21, 28, 43, and 50 line modes. Configurable keyboard.

Price $49 plus $4 shipping and handling. CA residents add sales tax. 30 day money-back guarantee.

Whitney Software, Inc.
P.O. Box 4999
Walnut Creek, CA 94596
415-933-9019

RS# 298

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DO IT YOURSELF!

XenoCopy-PC

PC-DOS program lets your PC
Read/Write/Format
over 350 formats **$79.95**

XenoFont

high quality text screen printouts
$49.95

+$5.00 S/H + Tax if CA.
XenoSoft ▪ (415) 644-9366
2210 Sixth Street,
Berkeley, CA 94710

Number **123** on the Reader Service Card

Network API Library

For *C* or *Pascal*, supports both *Microsoft* and *Turbo* compilers, Compatible with *IBM, 3Com, Banyan*, and *Novell* Networks. Includes DOS network functions, **NetBIOS, IPX/SPX, Client/Servers, Remote Procedure Calls,** Remote screens/windows. $99.95

NetWorkStation Utilities

Network Consulting, Distributed Processing, Classroom Screen Broadcasts, Work Groups, Device Sharing. NetBIOS/Netware $99.95

Information Software Associates
Box 2582, Iowa City, IA 52244
Call for more info **(319) 354-6804**

Number **294** on the Reader Service Card

PC TIMER TOOLS

PCHRT is the definitive answer to execution profiling and embedded timer applications in a PC/MSDOS environment. 40 functions manage multiple microsecond timers, generate precision delays, and insert timers in virtually any interrupt. Profile BIOS, DOS, and EMS effortlessly! Supports TC, TP, MSC. Libs, manual, examples, and full source included.
$49.95 ppd USA, elsewhere add $4.00. VISA/MC.

RYLE DESIGN
P.O. Box 22
Mt. Pleasant, MI 48804
517-773-0587

Number **210** on the Reader Service Card

IS 640K JUST NOT ENOUGH?

The **ATLAS MEMORY BOARD** adds 64K of DOS USABLE memory to the base 640K, making the total CONVENTIONAL memory 720,896 bytes. It does not use any memory resident drivers, and is compatible with all known applications. The same board works in any PC, XT, AT or 386 compatible, with monochrome graphics or CGA video. Memory added is NOT extended or expanded memory. The Atlas memory board has been certified by both NOVELL and 10 Net. Most orders shipped same day.

ATLAS MEMORY BOARD
15 Day Free Evaluation **$149.00** *Qty 1 Quantity Discounts Available*

M/C **1-800-535-6712** VISA
8AM - 9PM MON-SAT
NATIONAL DISCOUNT COMPUTERS
702-459-0049 NV Residents & Tech Support

Number **107** on the Reader Service Card

Why you want BATCOM!

BATCOM is a batch file compiler that compiles your ".bat" files to ".exe" files to make them faster, more professional, and more capable. **BATCOM** extends DOS with new commands so you can read keyboard input, perform arithmetic, use subroutines, and much more. In addition, **BATCOM** protects your source code, and you can distribute your compiled programs without royalties. Only **$59.95**

ORDER TODAY!

Wenham Software Co.
5 Burley St.
Wenham, Ma. 01984
(508) 774-7036

Number **320** on the Reader Service Card

QuickBASIC File Indexing

Creating B+ Tree indexes in QuickBASIC with **Index Manager™**. Access files randomly by full or partial key, or sorted forward or backward. One module performs all functions making **IM** very easy to use. Only indexes managed! You retain full control over your data. Assembler code and cache buffers make **IM** very fast. Demo on GEnie and CompuServe. Eight sample programs and 64 page Users Guide included.
Only $59 (+tax in CA). VISA/MC accepted.

CDP Consultants
1700 Circo Del Cielo Drive
El Cajon, CA 92020
(619) 440-6482

Number **335** on the Reader Service Card

PC-lint

C Programmers: Locate bugs, glitches, quirks and inconsistancies in your C code. PC-lint supports multiple modules, K&R, ANSI. Specify OS/2 or MS-DOS. Only $139.

Gimpel Software
(215)584-4261
3207 Hogarth Ln
Collegeville PA 19426
30 Day Money-back Guarantee

Please contact this advertiser directly.

TURBO PASCAL PROGRAMMERS

TP3, an easy-to-use preprocessor, permits usage of assembly language rather than in-line statements to generate in-line machine code. The assembly source is retained as commentary.

All 8086 and 8087 instructions are supported using MASM syntax. Pascal source passes through unchanged.

A really useful addition to your utility library. 5.25" ds/dd (xt) floppy. $49

MAIKON SOFTWARE
P.O. Box 24467
San Jose, CA 95154
(408) 223-1166

Number **354** on the Reader Service Card

The EDitor

Powerful **UNIX**-style programmer's editor. For use on IBM or compatible. Comprehensive manual included. A solid value at only $74.95 (US).

Available from:

Blue Jay Systems
PO Box 608
Mount Vision, NY 13810
607 293-8007

Number **11** on the Reader Service Card

GRAF/DRIVE PLUS

Turbo C/Pascal Graphics Drivers for Printers and Plotters

LaserJet–Epson–HP7470. Just plug our BGI-compatible drivers into Turbo's *initgraph*.

- Port your screen graphics program to hard copy devices with only minor changes.
- Get the full resolution of the output device, not a screen dump.
- Not TSR, Max 16K.

Personal $149. No-royalty developer $299. 30-day m/b guarantee.

FLEMING SOFTWARE
PO Box 528 Oakton, VA 22124
(703) 591-6451

Number **178** on the Reader Service Card

Job-hunting in Microsoft® Country?

1990 Northwest Computer Industry Employment Directory

$39.95. From the publisher of the annual guide ***Northwest High Tech*** (book and database).

"I recommend it for anyone serious about career opportunities in the Pacific Northwest."
—John C. Dvorak
syndicated columnist

Resolution Business Press
11101 N.E. 8th St., Suite 208
Bellevue, WA 98004
(206) 455-4611

Number **38** on the Reader Service Card

80386 MACRO DISASSEMBLER

MASTERFUL DISASSEMBLER is the worlds first macro disassembler. Interactive by design not after thought. Features:

- User defined macros with arguments
- Auto code/data separation
- Auto commenting and label naming
- Command menus and help screens
- Customizable disassembly options
- Comprehensive, indexed, 100 pg manual
- Fully supports 8086/87 thru 80386/87
- COM/EXE/SYS or other files or memory
- Cross reference and much much more!

Still only $67.50 + tax ($1.50 s/h).

C.C.SOFTWARE
1907 ALVARADO AVE.
WALNUT CREEK, CA 94596
(415)939-8153

Number **132** on the Reader Service Card

Coming Next Issue:

Programming in the Protected Mode

July/August 1990 Vol. 8.4

Advertising Space Reservation Deadline:
May 4, 1990
Materials Due:
May 11, 1990
Please call ***Programmer's Journal*** to reserve your space.
503-747-0800

Issue Special

Code Disks

$8.00
Reg. $10.00
See page 88 for details.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Dup-Tek

SOFTWARE DUPLICATION
High Quality — Low Prices
Fast Turnaround

- Guaranteed diskettes for sale
- Rigid quality controls
- 5.25" duplication as low as $.55 each
- Label production & application
- 100% satisfaction-guarantee policy

VISA — MASTERCARD — C.O.D.

DUPLICATION TECHNOLOGIES LTD.
Route 1, Box 113
Wellington, MO 64097
816-934-8518 FAX: 816-934-2617

Number **344** on the Reader Service Card

The $25 Network

Try the 1st truly low cost LAN

- Connect 2 or 3 PCs, XTs, ATs, PS/2s
- Uses serial ports and 5 wire cable
- Runs at 115K baud, up to 90 feet
- Transfer 8500 bytes per second (ATs)
- Runs in background, totally transparent
- Share any device, any file, any time
- Needs only 14K of ram
- Replace all file transfer software
- $25 per network, NOT PER NODE!
- OVER 15,000 SOLD WORLD WIDE

Skeptical? We make believers!

iModes Information Modes
Drawer F, Denton, TX 76202
817-387-3339 Technical
1-800-628-7992 Orders

Number **348** on the Reader Service Card

OPT-TECH SORT/MERGE

Extremely fast Sort / Merge / Select utility. Run as an MS-DOS command or CALL as a subroutine.
Supports most languages and filetypes including Btrieve and dBase. Unlimited filesizes, multiple keys and much more!

MS-DOS **$149.**
OS/2, UNIX **$249.**

Opt-Tech Data Processing
P. O. Box 678
Zephyr Cove, NV 89448
(702) 588-3737

Number **16** on the Reader Service Card

Protect Your Software and Your Profits.

Encrypted key-interrogation routines and custom wired hardware provide the highest degree of security available. Our devices can also be used for serialization, leasing, revenue collection, demo control, etc. Please write or call for more information or a demonstration package.

PROTECH
MARKETING, INC.
9600-J Southern Pines Blvd.
Charlotte, NC 28217
800-843-0413
704-523-9500 Fax 704-523-7651

Number **228** on the Reader Service Card

sPORTt
serial PORT tester

sPORTt is a serial port tester for people who don't want to be data communications experts, but use or write software for serial devices, such as plotters, mice, digitizers, modems, serial printers, or computer to computer links.

- displays all data going in and out in ASCII, decimal, hex, binary
- lets you choose handshaking lines
- sends/receives disk files
- automatically runs port diagnostics

Order today only $89.95!

WHS
3037 Grass Valley Hwy #8201
Auburn, CA 95603
916-885-2480

Number **329** on the Reader Service Card

SHAREWARE AUTHORS

Give your programs the chance they deserve! With Megapost, your program is uploaded to CompuServe, GEnie, Delphi, BIX, and EXEC-PC. The low fee is probably less than you would pay just to open accounts with all of these services. Included are one half-price update and notification of the downloading procedure and 3-month download count for each service. 15% discount for ASP members.

Write for a free brochure.

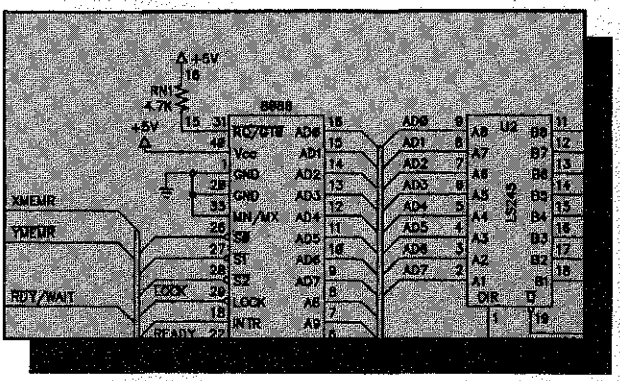
Andrew M. Saucci, Jr.
641 Koelbel Ct.
Baldwin, NY 11510-3915

Number **347** on the Reader Service Card

Offer your product directly to Professional Programmers:

Placing an ad in **PJ BILLBOARD** is an easy, low-cost way to reach potential buyers. Regular listings include a headline, company information, and approximately 400 characters for your message. Display ads are 1 5/8" wide by 2 3/8" high. For more information or to reserve a space, call our ad department at **503-747-0800.**

WALL-SIZE XT SCHEMATIC



At last you can plumb the mysteries of your computer with this single sheet (24" x 36") schematic of the IBM XT's main board. This is a wealth of information for both True Blue *and* clone owners.

This is the kind of information you need for tracking down board problems, for designing add-on cards, or for turning XT motherboards into inexpensive controllers/monitors. And, because this is a single-sheet schematic, you won't spend hours tracking signals from page to page. You see the entire motherboard at once.

Trouble-Shooting Information

As a bonus, we include trouble-shooting information. For instance, your board is dead but a quick look at the schematic shows you the location of PPI port A. You know that the power-on self test loads a number into port A each time it completes a self test. All you need to do is read the port with a logic probe to see how far the system went before it discovered an error. We include these checkpoints and other trouble shooting information with the schematic.

NOTE: Although clones often differ a bit from IBM in layout and IC designations, their circuits are close enough to the original XT for this schematic to be very useful.

IBM PC-XT Schematic $15.00
U.S. shipping and handling $1.95
Foreign (airmail shipping and handling) $3.95

THE MICRO TECHNICAL JOURNAL
MICRO CORNUCOPIA

Dept PJ, P.O. Box 223, Bend, Oregon 97709

Monday-Friday, 9AM-5PM PST
503-382-5060
FAX: 1-503-389-6833 (24 hours)

PROTECH
MARKETING, INC.

VISA MC

Contact this advertiser directly.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ADVERTISER INDEX

Reader Service	Advertiser Name	Page Number
305	Acme Software	63
345	Alphax Systems, Inc.	36
30	The Austin Code Works	39
207	Automation Education	45
51	Az-Tech Software, Inc.	93
343	Base Two Development	65
2	Blase Computing Inc.	71
15	Blaise Computing Inc.	73
14	Blaise Computing Inc.	75
244	Block Island Technologies	16
11	Blue Jay Systems	94
3	Borland International	C4
288	Burton Systems Software	93
132	C. C. Software	94
321	Catenary Systems	54
335	CDP Consultants	94
340	Computer Books	93
300	Computer Language Arts	14
229	Disk Software Inc.	46
344	Duplication Tech Ltd.	95
316	EDI	C3
349	Eikon Systems	45
353	EMS	93
352	EZX	93
178	Flemming Software	94
261	Genus Microprogramming	32
355	Geomaker Software	75
*	Gimpel Software	94
18	Goin Places Software	86
334	Hamilton Laboratories	46
42	Harvard Softworks	83
348	Information Modes	95
294	Information Software Associates	94
358	Inspired Innovations	14
346	JDyx Enterprises	20
295	Knowledge Dynamics Corporation	87
155	Komputerwerk Inc.	93
63	Laboratory Microsystems Incorporated	92
44	Lahey Computer Systems Inc.	75
*	Logical Alternatives Incorporated	93
354	Maikon	94
223	Marietta Systems Inc.	93
291	MBAC	93
*	*Microcornucopia*	95
104	Microway	11
121	MISOSYS, Inc.	17
266	Mix Software	71
94	MMC AD Systems	52
312	MTD Software Development Co. Ltd.	85
15	Multiscope Inc.	44
109	Nanosoft	36
107	National Discount Computers	94
278	Netwise	69
160	Nu-Mega Technologies	3
16	Opt-Tech Data Processing	95
*	Paradigm Systems	85
208	PC Tech, Inc.	20
6	The Periscope Company, Inc.	37
27	Pinnacle Publishing Inc.	47
*	***PJ* Code DiskScription**	8
279	Pocket Soft, Inc.	9
*	***Programmer's Journal* Mail Order Store**	88
114	Programmer's Shop	76–77
228	Protech Marketing Inc.	95
252	Quantasm Corporation	92
147	Quarterdeck Office Systems	C2
296	Que Corporation	89
34	R Fringe Publishers Inc.	83
20	Rational Systems Inc.	16
38	Resolution Business Press	94
210	Ryle Design	94
26	Sage/Polytron Corporation	12–13
347	Saucci, Andrew M., Jr.	95
269	Sequiter Software Inc.	93
98	The Software Engineering Store	68
356	Software Projects Group	32
198	Software Science Inc.	78
221	Software Security, Inc.	5
110	Softway, Inc.	24
47	Solution Systems	1
10	South Mountain Software	38
330	Stanwood Associates Inc.	26
158	Summit Info Systems, Inc.	37
289	TechnoJock Software, Inc.	87
166	The Tool Makers	38
327	Tools & Techniques Inc.	14
276	Tri-Technology Systems Inc.	17
25	TurboPower Software	85
309	UnderWare, Inc.	53
341	V Communications, Inc.	63
125	V Communications, Inc.	70
328	Vermont Database Corporation	93
304	Visitech Software	87
320	Wenham Software Co.	94
100	Western Wares	17
298	Whitney Software, Inc.	93
329	WHS	95
123	XenoSoft	94

When contacting any of the above advertisers, please tell them you saw their ad in ***Programmer's Journal*** and that you appreciate their continuing support of this magazine. The advertisers with an * prefer to be contacted directly. ***PJ***'s Advertiser Index is provided as a free service, and ***Programmer's Journal*** cannot be held responsible for omissions or errors. ***Programmer's Journal*** is not responsible for errors in advertisements.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A message from "the Author's Network"

You want high quality software, right? But you don't want to take a loan out to buy it. And you want it now. Then you should know about an exciting, new on-line service. It's called **EDI Author-Direct** and it's going to change the way you think about software.

EDI Author-Direct was created by software authors. On-line fees collected from users like yourself pay for the software you download. The result - you pay less for the software you download. How much less? In most cases you pay only for your on-line time, which may vary between 17 and 20 cents per minute (depending on the data network you choose to access us).

EDI Author-Direct is packed with great programs. Download a copy of **Sagewords**, J. Keith Phipps's word-processing program. It can be yours in minutes! Or how about William Halls' **Commo**, a fully developed communications program. Interested in protecting your computer from viruses? Get a copy of David Leithauser's **Virus Stopper II**. Need a mailing list program? Download **Label Master Version 4.0**. We use it ourselves. You name it, you can find it on EDI Author-Direct.

Since we are an author-direct network, you get the latest product releases direct from the author. If you have a problem, get back on-line and talk direct with the author. On-line forums put you in touch with authors so your questions can be answered quickly and accurately.

To get connected call one of the *Minitel Data Network* access numbers below and follow the instructions provided. If you cannot find a phone number in your immediate area, hook up with us via **BT TYMNET** (formally TYMNET), one of the nations premier data networks with local access numbers in over 850 cities and eighty countries worldwide. We will be on-line with BT TYMNET by mid March. For your local access number, call BT TYMNET customer support (24 hours a day) at 1-800-336-0149.

ACCESS INSTRUCTIONS (*Minitel Data Network*): Set your modem to 7-E-1 and call one of the numbers below. Once connected press RETURN until you see a # prompt. Then press **x** and RETURN to get a star prompt. Type .vtt (remember the period) to gain network access. Put in your credit card info to pay for the call and type EDI to access us on the network. On-line costs are 17 cents per minute, even at 2400 baud. There are no start-up costs. Prices subject to change without notice.

ALABAMA
Anniston (205)820-7609
Birmingham (205)254-3120
Montgomery (205)279-0314
ARIZONA
Phoenix (602)233-9592
ARKANSAS
Little Rock (501)376-1535
CALIFORNIA
Burbank (818)954-9161
Corona (714)736-6820
El Monte (818)442-9291
Garden Grove (714)895-4946
Los Angeles (213)772-6971
Monterey (408)899-7855
Mountain View (415)964-8240
Newport Beach (714)895-4946
No. Hollywood (818)786-4111
Sacramento (916)452-7356
San Bernardino (714)888-7611
San Diego (619)293-3620
San Fransisco (415)391-9080
San Jose (408)554-0344
Van Nuys (818)786-4111
COLORADO
Colorado Springs (719)540-8220
Denver (303)893-3803
CONNECTICUT
Hartford (203)728-6766
Milford (203)728-6766
New Haven (203)877-2751
Stamford (203)327-6745
Wilmington (302)571-1433
D.C.
Washington (301)937-0530
Washington (301)937-0722
FLORIDA
Fort Lauderdale (305)462-3104
Jacksonville (904)398-2074
Miami (305)651-2387
Pensacola (904)438-7203
Tampa (813)626-5235
GEORGIA
Atlanta (404)256-9151
Augusta (404)790-5406
Columbus (404)682-1103
HAWAII
Honolulu (808)524-7950
ILLINOIS
Chicago (312)704-0313
Oak Brook (312)572-8550
Oak Brook (312)572-8566
Rock Island (319)324-0042
Springfield (217)525-6683
INDIANA
Indianapolis (317)634-4301
IOWA
Davenport (319)324-0042
Des Moines (515)224-0011
KENTUCKY
Louisville (502)587-9560
LOUISIANA
New Orleans (504)566-0002
MARYLAND
Baltimore (301)760-1950
MASSACHUSETTS
Boston (617)965-4900
MICHIGAN
Detroit (313)963-9141
Grand Rapids (616)456-6047
MINNESOTA
Minneapolis (612)338-5540
MISSOURI
Kansas City (816)283-3719
St. Louis (314)721-4801
NEBRASKA
Omaha (402)341-2425
NEW JERSEY
Boonton (201)335-2332
Cherryhill (609)235-3220
Hackensack (201)489-9020
Moorestown (609)235-3220
New Brunswick (201)846-9310
Newark (201)564-6300
Parsippany (201)335-2332
Springfield (201)564-6300
Trenton (609)883-5113
Whippany (201)335-2332
NEW YORK
Buffalo (716)896-5410
Long Island (516)481-8028
New York (212)391-1920
New York (212)391-0370
Poughkeepsie (914)454-1808
Purchase (914)694-6842
Syracuse (315)428-1674
NORTH CAROLINA
Charlotte (704)374-0587
Fayetteville (919)436-0145
Raleigh (919)821-7535
NORTH DAKOTA
Fargo (701)237-5282
OHIO
Akron (216)535-6635
Cincinnati (513)241-1446
Cleveland (216)621-3880
Columbus (614)461-8450
OKLAHOMA
Lawton (405)355-0275
Oklahoma City (405)272-9751
OREGON
Portland (503)241-0549
PENNSYLVANIA
Harrisburg (717)236-5241
Philadelphia (215)829-0912
Pittsburgh (412)261-9305
SOUTH CAROLINA
Columbia (803)771-7665
SOUTH DAKOTA
Sioux Falls (605)334-1832
TENNESEE
Memphis (901)521-0529
Nashville (615)292-6801
TEXAS
Austin (512)499-8029
Dallas (214)638-7850
Fort Worth (817)460-3711
Houston (713)981-9797
Kileen (817)532-8980
San Antonio (512)525-1692
UTAH
Salt Lake (801)544-0086
VIRGINIA
Newport News (804)874-5121
Norfolk (804)858-2715
Richmond (804)788-0401
WASHINGTON
Seattle (206)624-3580
Spokane (509)455-9398
WEST VIRGINIA
Charleston (304)345-7505
Parkersburg (304)485-7569
WISCONSIN
Madison (608)255-5857

ELECTRIC DISTRIBUTION
INTERNATIONAL

Coming March 15th, BT TYMNET Access. Call 1-800-336-0149 for your local access number and type SOFTWARE at the login prompt.

1332 Hermosa Avenue, Suite 7, Hermosa Beach, CA 90254. (213) 374-3478, 800-676-6076

Number 316 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

BORLAND'S TURBO C PROFESSIONAL 2.0

CORPORATE TOP RATED PC WEEK 1989 SATISFACTION POLL

PC WEEK POLL: C COMPILERS

	Overall Weighted Score	Overall Reliability	Complete of Command Descript.	Overall Perform.	Complete & Organiz. Document.	Document Clarity	Compiling Process Efficiency	Product Support Quality	Value Relative To Cost	Product Support Access.
Turbo C 2.0 (Borland International)	81	87	79	84	77	78	86	72	70	93
C Optimizing Compiler 5.1 (Microsoft Corp.)	76	83	80	81	78	74	76	68	67	70
C++ 1.07 (Zortech Inc.)	66	68	64	71	63	63	69	60	58	76

"Microsoft was No. 1, but they have been unseated by Borland." PC Week, *May 8, 1989*

PC WEEK POLL: SOFTWARE DEBUGGERS

	Overall Weighted Score	Overall Reliability	Effective. Programmer Interface	Document. Clarity	Complete. Command Descript.	Complete. & Organize. Document	Overall Perform.	Integration Within Programming Environment	C Compiler Compatibility	Product Support Quality	Product Support Access	Value Relative To Cost
Turbo Debugger 1.0 (Borland International)	84	89	90	81	81	81	89	88	81	73	72	93
Codeview 2.2 (Microsoft Corp.)	73	80	71	72	74	74	74	74	78	67	64	72

"Borland's Debugger outshines Microsoft's Codeview." PC Week, *May 15, 1989*

It's two winners in one.

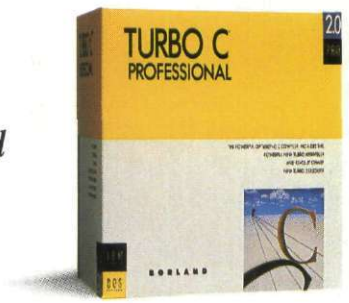
Turbo C,® the core of Turbo C Professional, was the outright winner in *PC Week*'s Poll of Corporate Satisfaction on C compilers. Overall, Borland won with 81. Microsoft® placed second.

Turbo Debugger,® also included in Turbo C Professional, was the outright winner in EVERY category in *PC Week*'s Poll Of Corporate Satisfaction on Debuggers. And, once again, we topped the score with 84, overall. Microsoft came in second-best, 11 points behind.

Get Borland's Turbo C Professional and get the best of both worlds: our top-rated C compiler and our top-rated Debugger.

Call **(800) 345-2888*** and we'll send you both *PC Week* polls and technical specifications on Turbo C and Turbo Debugger.

Turbo C Professional includes both Turbo C 2.0 and Turbo Assembler® & Debugger.



BORLAND

MC 61

From Canada, Call (408) 848-4391. Reprinted from PC Week, May 8, 1989; PC Week, May 15, 1989. Copyright ©1989 Ziff Communications Company. Turbo C, Turbo Debugger, and Turbo Assembler are registered trademarks of Borland International. Copyright ©1989 Borland International, Inc. All rights reserved. BI 1328

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED